Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Jasse Lahdenperä

Design and Implementation of a Virtual Reality Application for Mechanical Assembly Training

Master's Thesis
Espoo, October 29, 2019

Supervisor: Assistant Professor Yu Xiao
Advisor: Postdoctoral Researcher Ji-Hye Lee

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Jasse Lahdenperä

Title:
Design and Implementation of a Virtual Reality Application for Mechanical Assembly Training

Date: October 29, 2019 **Pages:** 72

Major: Mobile Computing, Services and Security **Code:** SCI3045

Supervisor: Assistant Professor Yu Xiao

Advisor: Postdoctoral Researcher Ji-Hye Lee

Although virtual assembly has been studied for over 20 years, it has not yet reached a state where it would enjoy widespread usage outside of academia despite the possible cost savings and improvements in the effectiveness of the training. Even though there have been multiple separate studies on virtual assembly, hand-based interaction, and assembly assistance, we have not found applications that would combine all of these to provide a complete assembly training experience.

The goal of this thesis was to design and implement a virtual reality application for mechanical assembly training. In our application, we provide a natural user interaction by using a Leap Motion controller, a hand tracking device mounted onto a virtual reality headset. The application was implemented using the Unity game engine and supports both Oculus and SteamVR compatible VR headsets.

Unlike most of the previous systems, we combine the use of hand-based interaction, assembly simulation, and context-aware assembly guidance to create an all-in-one VR assembly solution. As a part of our implementation, we propose a new method for assembly guidance and validation that works by matching assemblies built by the user to the assembly the user is supposed to build.

Based on the user testing results, there is an interest in this kind of application. Although the inaccuracies with the hand and finger tracking hindered the usability of the application, the users described the application as surprisingly easy to use once they learned how to overcome these issues.

Keywords: virtual assembly, virtual reality, mechanical assembly, assembly training, Unity, Leap Motion

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Jasse Lahdenperä

Työn nimi:
Mekaanisen kokoonpanon harjoitteluun tarkoitetun virtuaalitodellisuuteen perustuvan ohjelman suunnittelu ja toteutus

Päiväys:	29. lokakuuta 2019	**Sivumäärä:**	72
Pääaine:	Mobile Computing, Services and Security	**Koodi:**	SCI3045
Valvoja:	Apulaisprofessori Yu Xiao		
Ohjaaja:	Tutkijatohtori Ji-Hye Lee		

Siitä huolimatta, että virtuaalista kokoonpanoa on tutkittu yli 20 vuotta, ja se voisi tarjota sekä kustannussäästöjä että jopa parantaa harjoittelun tehokkuutta, se ei ole vielä saavuttanut vakiintunutta asemaa akateemisen tutkimuksen ulkopuolella. Vaikka virtuaalisesta kokoonpanosta, käsipohjaisesta vuorovaikutuksesta ja kokoonpanon avustamisesta on tehty useita erillisiä tutkimuksia, emme ole löytäneet sovelluksia, jotka yhdistäisivät kaikki nämä kokonaisvaltaisen harjoitusalustan tarjoamiseksi.

Tämän diplomityön tarkoituksena oli suunnitella ja toteuttaa virtuaalitodellisuuteen perustuva työkalu mekaanisen kokoonpanon harjoitteluun. Ohjelmamme tarjoaa luonnollisen, käsien seurantaan perustuvan käyttöliittymän hyödyntämällä virtuaalilaseihin kiinnitettyä Leap Motion -ohjainta. Sovellus toteutettiin käyttäen Unity-pelimoottoria ja sovellus tukee sekä Oculus- että SteamVR-yhteensopivia virtuaalitodellisuuslaseja.

Toisin kuin useimmat vastaavat järjestelmät, meidän työkalumme yhdistää käsin tapahtuvan interaktion, kokoonpanosimulaation ja kontekstisidonnaiset kokoonpano-ohjeet tarjoten kokonaisvaltaisen sovelluksen virtuaalisen kokoonpanon harjoitteluun. Osana työkaluamme kehitimme uuden menetelmän kokoonpanon aikana tapahtuvien virheiden havainnoimiseen ja kontekstisidonnaisten kokoamisohjeiden muodostamiseen. Kehittämämme menetelmä perustuu vastaavuuksien etsimiseen käyttäjän kokoamien tuotteiden ja tavoitteena olevan tuotteen väliltä.

Käyttäjätestauksesta saatujen tulosten perusteella tämänkaltaiselle sovellukselle olisi kysyntää. Vaikka käsienseurantalaitteen epätarkkuus haittasi sovelluksen käytettävyyttä, käyttäjät luonnehtivat sovellusta yllättävän helppokäyttöiseksi opittuaan työskentelemään sovelluksen parissa.

Asiasanat:	virtuaalinen kokoonpano, virtuaalitodellisuus, mekaaninen kokoonpano, kokoonpanon harjoittelu, Unity, Leap Motion
Kieli:	Englanti

Acknowledgements

First, I would like to thank Assistant Professor Yu Xiao for supervising this master's thesis and also for instructing in my bachelor's thesis already in 2016. I would also like to thank my advisor, Ji-Hye Lee, for her valuable feedback during the implementation of the application and also for helping me with the organisation of user testing. I also thank all the testers who participated in the user test for their valuable feedback and ideas for future improvements.

This thesis would have been much more difficult to produce without my experience in the games industry and working with Unity. Therefore, I would also like to thank Rovio Entertainment for not only offering me an exciting career but also for being flexible in regards to studying on the side of my daily work. While managing time between a full-time job and full-time studies has sometimes been tricky, I have truly enjoyed the last six years.

Finally, I would like to thank my fiancée, Suvi. While the last six years have definitely been busy every once in a while for both of us, you have helped tremendously in maintaining a healthy(ish) work-life balance.

Espoo, October 29, 2019

Jasse Lahdenperä

Abbreviations and Acronyms

CAD	Computer Aided Design
CBM	Constraint-Based Modelling
CCD	Continuous Collision Detection
DOF	Degrees of Freedom
FOV	Field of View
FPS	Frames Per Second
HCI	Human-Computer Interaction
HMD	Head-Mounted Display
IMU	Inertial Measurement Unit
PBM	Physics-Based Modelling
SDK	Software Development Kid
VA	Virtual Assembly
VCG	Virtual Constraint Guidance
VR	Virtual Reality

Contents

Chapter 1

Introduction

Training new employees for mechanical assembly tasks can be both time-consuming and expensive as it often requires someone more experienced to teach the basics of the assembly task. Additionally, while physical training with a more skilled teacher has been proven to work in real life, some earlier studies such as [1, 17, 41] have shown promising results of assembly training in virtual environment being more productive than corresponding physical training. Furthermore, virtual reality-based training has other added benefits such as possibilities to train wherever, whenever and without supervision. For these reasons, VR-based assembly training provides an attractive alternative to traditional training methods.

Although virtual assembly (VA) has been a topic of broad academic interest for over 20 years, its practical applications in the industry are still limited. We believe this is due to the following reasons: First, most of the existing VA applications have been controlled using unnatural user interfaces such as wands, or pen-like input devices. Second, while multiple studies have worked on making the operation of connecting two virtual objects more dynamic with the use of physics and kinematic constraints, the physical behaviour has been widely neglected in more complex assembly simulations. Therefore, existing assembly simulations have been dull, non-physical simulations that do not match the real-life experience of performing the same operation.

The goal of this thesis was to design and implement a virtual reality application for mechanical assembly training that would address the issues mentioned above by providing a natural user interface combined with a dynamic assembly simulation where parts can physically interact with each other. Using readily available virtual reality technology and a commercially available game engine, we developed an application that utilises hand tracking to provide a natural way of interacting with the virtual environment and assembly pieces. To create an assembly simulation that is both physically

convincing and hand tracking friendly, we utilise a combination of real-time physics for natural interactions between objects and kinematic constraints for easier assembly operations. In addition to previously mentioned challenges, we also wanted to address the problem in many existing virtual assembly applications where the user is forced to follow the instructions in a fixed order even though it would be possible to complete the task in alternative ways. To solve this, we propose a new method for assembly validation and guidance that works by matching the assembly built by the user to the target assembly. With this new method, we provide the user with contextual assembly instructions that automatically detect what the user has built so far, and update according to that rather than relying on a fixed assembly sequence.

This thesis was done as a part of Cognitive Engine for Assembly and Maintenance Automation (CEAMA) -project and the primary methodologies used were a literature review, experimenting, and iterative development based on the results of user tests. We tested our application in two separate user tests with university staff and professionals working on the engineering industry. Based on the user testing results, there is an interest in this kind of an application. Although the accuracy of the hand tracking hindered the usability of the application, the users found the contextual assembly instructions to be easy to follow, and many of them described the application as surprisingly easy to use after getting accustomed to the controls.

The rest of this thesis is organised as follows: Chapter 2 offers insight into virtual reality and previous studies regarding virtual assembly and assembly theory. Chapter 3 explains the high-level design of the application. Chapter 4 goes through the implementation of the application and the problems encountered during the development. Chapter 5 focuses on the evaluation of the application. Finally, Chapter 6 concludes the thesis and presents future work.

Chapter 2

Background

This chapter introduces virtual reality and technologies relevant to the topic of the thesis. In Section 2.1, we briefly introduce virtual reality and related hardware. Section 2.2 offers insight into previous studies in virtual assembly, virtual hand-object interaction, and assembly theory.

2.1 Virtual reality overview

The definition of virtual reality varies from source to source, and it is, therefore, hard to get right. As a good starting point, Sherman *et al.* defined the fundamental elements of virtual experience as virtual world, immersion, sensory feedback, and interactivity [44]. This set of elements reflects quite well the experience modern virtual reality solutions can provide us, and in this thesis, we stick to this definition.

While the history of VR extends back to Morton Heilig's *Sensorama*, a non-interactive virtual reality setup in 1956 [44], it has mainly been a topic of academic interest until the recent technological advancements that have finally brought affordable VR headsets available for customers. VR entertainment, primarily VR gaming, has been one of the driving forces of VR development during recent years. Since the software development kit (SDK) of Oculus Rift was released, several new game companies have surfaced, focusing solely on VR gaming. Additionally, many existing gaming companies have also started thinking about making their existing games at least partially VR compatible.

A typical modern VR experience satisfies the aforementioned fundamental elements with the use of specialised output devices (Section 2.1.2) such as head-mounted displays and 3D-enabled controllers (Section 2.1.1) like handheld wand controllers. Together these devices allow the user to become im-

mersed in the virtual world and interact with it.

Due to its versatility, VR has been widely adapted to different applications across multiple fields. Some of the most notable use cases include VR gaming, vehicle simulators such as flight simulators, applications for information visualisation, and medical applications such as stroke rehabilitation [51]. Besides, there have been some recent attempts in combining VR with gamification, the idea of applying game design elements to non-gaming application, as there have been attempts to apply game design elements to, for example, fitness applications to encourage exercise. Virtual reality-based assembly training and assembly simulation have also been a topic of broad academic interest throughout the years and is discussed further in Section 2.2.1.

2.1.1 User interfaces

Human-computer interaction (HCI) focuses on the interface between people and computers. Since interaction is one of the key elements of VR experience, the interaction between the user and the computer plays a significant role in a VR experience. Although it is possible to use traditional keyboard and mouse input in VR, it is less common due to modern VR focusing on the use of head-mounted displays (HMD) with orientation and position tracking (Section 2.1.2) where conventional computer input would significantly limit the usability and weaken the immersion. For this reason, multiple different user interfaces have been suggested for modern VR hardware.

Wand controllers are handheld controllers that are used to interact with the virtual world by pointing the device towards virtual objects to manipulate them. While wand controllers have been widely used in earlier research, Nintendo Wii Remote [15] was the first mainstream wand controller, and its success has had a profound impact on the development of modern game controllers. Another popular wand controller PlayStation Move [38], followed Wii Remote soon and nowadays most of the conventional VR controllers are wand controllers. Most of the wand controllers track either orientation or position and orientation of the controllers providing either 3 or 6 degrees of freedom (DOF) tracking, correspondingly. Both headsets used during development, Oculus Rift [35] and HTC Vive [49] also come with their wand-style controllers that can be used to point at things, and they also provide plenty of different actions through additional buttons in controllers. Oculus Touch controllers can even provide virtual hands for the user by taking advantage of multiple proximity sensors placed around the controllers.

Pen-like controllers, such as Virtuose ^{TM} 3D Desktop [48] have also been prevalent in academic studies since they are often accurate and capable of

providing haptic feedback. However, because of their high costs and cumbersomeness, they have yet to become popular outside academia and tech industry. Moreover, this kind of input devices are not well suited for modern VR hardware that revolves around the usage of HMDs where movement inside the virtual world is a crucial part of the experience.

Multiple attempts on developing VR gloves for natural interaction between the user and virtual objects have been made to improve the immersion in VR. These gloves usually utilise either inertial measurement units (IMUs) or flex sensors to track hand and finger movements. Although these approaches are mostly indistinguishable from users' point of view, many of the flex sensor-based gloves are limited to tracking flexion movement (bending) of the fingers, unlike the IMU-based gloves that can also track abduction (spread) of the fingers. In addition to commercial solutions, multiple studies such as [26, 52] have developed their own VR gloves.

Optical systems such as Kinect, Leap Motion, and uSense's Fingo allow user to interact with the environment using their hands without having to wear VR gloves or hold controllers in their hands. While some systems such as Kinect only support gesture controls, Leap Motion and Fingo provide quality hand and finger tracking when placed on a desktop or attached directly to the VR headset. Therefore, these devices can support more complex interactions, such as grasping and throwing objects. Although optical systems enjoy the benefit of leaving your hands free, their accuracy relies on unobstructed visibility between camera and tracked hands which means that any occlusion, such as another hand in front of the tracked hand, may significantly degrade the accuracy of the tracking. Moreover, optical tracking may impose some limitations on the environment as mirroring surfaces or bright lights can lower the tracking accuracy. Another significant disadvantage compared to the wearable systems is the lack of force feedback that has been shown to improve performance in virtual assembly [25, 53]. However, since optical systems are often cheaper than their wearable counterparts, their adaptation has so far been broader, and multiple studies have used Kinect or Leap Motion as user interface. In our application, we also chose to use Leap Motion because of its affordability, availability, and relatively good tracking accuracy.

While voice controls have been steadily getting into consumers' homes through smartphones, smart TVs and smart speakers, the use of voice commands in VR is still limited even though many of the HMDs come with integrated headphones and microphones. In addition to using Leap Motion for interaction with the world and controlling the application, we also utilise voice commands as an alternative option to controlling the application to get some insight of how the potential target audience would feel about using

voice controls.

2.1.2 Output devices

Since virtual world and immersion are one of the key elements of a VR experience, the output device is maybe the single piece of VR hardware that has the most significant impact on the quality of the experience. Output devices used in VR can be divided into two fields, monoscopic displays and stereoscopic displays.

In monoscopic displays, such as flat computer displays, the 3D environment is projected to a 2D plane using 3D projection. While it has been successfully used in visual applications such as 3D modelling and computer gaming, the view is somewhat equivalent of looking at surroundings with one eye closed, which reduces the quality of depth perception as the binocular cues that provide additional depth information are lost. This makes manipulating objects with 3 DOF or 6 DOF input devices harder, and research has shown that manipulating objects in 3D with 6 DOF input devices is slower when using a monoscopic display compared to using a stereoscopic display [8].

Stereoscopic displays, on the other hand, present different information for both eyes, which allows the eyes, together with the human brain, to provide the user with depth perception by creating the image from each eyes' perspective [8]. Stereoscopic displays work by either providing the image to both eyes simultaneously or by providing the image to both eyes at different times while blocking the visibility from the other.

Active shutter glasses are one example of technology where both eyes receive their information at different times. To achieve this, the glasses are synchronised with the refresh rate of the display and alternate between both eyes to let only a single eye at a time to see the display. Some of the most significant downsides with shutter glasses are that they limit the users' field of view (FOV) and that they often make the image slightly dimmer compared to when not using them. Secondly, to provide a good image quality, the user must watch the display from a suitable angle which also reduces their usefulness.

Another widely used stereoscopic display technique is passive polarisation, where the glasses work by having one lens with vertical polarisation and other lens with horizontal polarisation combined with a display device capable of producing filtered images for both eyes [44]. Compared to active shutter glasses, passive polarisation glasses require no power and usually provide better viewing angles. However, one of the downsides with polarising glasses is that these glasses get lower resolution image since both eyes are required

to share the display, but on the other hand, polarising 3D glasses are usually cheap, which is one of the reasons that for a long time they were used as the go-to technology in 3D cinemas.

Recently, head-mounted displays (HMD) have become the most prominent stereoscopic display devices in VR with the release of entertainment-oriented HMDs such as Oculus Rift, HTC Vive, and Valve Index. Typically, a modern HMD is a flat screen mounted in front of the users' eyes accompanied with additional hardware such as headphones to enhance the VR experience. To keep the size and weight of the HMDs reasonable, HMDs typically use pair of Fresnel lenses to widen the FOV produced by the small, lightweight display. Since human eyes cannot comfortably focus on something too close, the lenses also help reduce eye strain as they allow placing the display further away from the eyes while still maintaining a reasonable size of the display panel. By mounting the display directly in front of users' eyes, HMDs are capable of covering users' FOV in a way that the user can only see the virtual environment which adds to the immersion in the VR experience which in turn leads to enhanced VR experience.

In addition to the display, modern HMDs usually come with additional hardware for tracking the orientation and position of the headset. While the orientation tracking gives the user the ability to look around the virtual environment, the positional tracking allows the user to move inside the virtual environment while having their hands free for other actions. Naturally, this is not possible with active shutter glasses or polarising glasses as the display is stationary and the glasses merely filter the image coming from the display. Furthermore, even in stationary actions happening while seated, the position tracking significantly improves the VR experience since humans tend to unknowingly induce small head movements to gain more information about the surroundings. Although the position tracking can improve the VR experience, in practice the user still needs additional means of moving within the virtual environment as their movement is limited by the physical environment around the user and the tracking capabilities of the headset. Usually, the position and orientation tracking is achieved by fusing the sensor data from multiple sources both inside the headset and outside the headset. For example, Oculus Rift tracks the user by using cameras connected to the computer that see infrared LEDs in the HMD. HTC Vive, on the other hand, uses photodiodes attached to headset to interpret signals from Vive lighthouses to determine the position and orientation of the headset. Furthermore, HMDs also use IMUs as another data source to obtain more accurate estimations about the position and orientation.

2.2 Related work

2.2.1 Virtual assembly

Defining virtual assembly can be a difficult task as the definitions varies slightly between different sources. When we are talking about VA, we refer to the following definition by Xia *et al.*:

> Utilizing VR technology, computer graphics, artificial intelligence, assembly theory and method, to construct the virtual model of the product and the virtual environment of the assembly layout, and then interactively analyse and simulate the product design result and assembly operation process. [53].

Even though the use of VR for product assembly has been studied for over 20 years, the practical industrial applications have yet to become widespread, although its potential has been acknowledged and multiple examples of potential use cases can be found including assembly design [19], assembly training [1, 2, 5, 27, 29, 33], maintenance training and quality control [33].

In addition to potential cost savings associated with virtual training, multiple studies have also found out that VR training can be more effective when comparing the trainees' performance to one trained with traditional training [1, 11, 17, 41, 54]. Although the factors causing this performance increase are still somewhat unclear, it is believed that enhanced interactivity can be partially attributed to enhanced interactivity and stereoscopic display [41]. The importance of immersive experience using stereoscopic display has also been backed up by Boud *et al.* [11] and Dwivedi *et al.* [17]. One contributing factor that makes non-immersive VR training less effective can also be that past research has highlighted that users have more success in positioning objects in a 3D environment when using a stereoscopic display [8].

VA can be roughly divided into two commonly used methods, physics-based modelling (PBM) and constraint-based modelling (CBM) [41, 53]. In the former method, objects interact physically with each other and connections between parts are formed by the physical contact between parts. In CBM, however, the assembly is performed by introducing physics constraints that reduce the degrees of freedom between the objects being manipulated, thus restricting the relative movement and rotation between the parts.

Physics-based modelling Although physics-based modelling can provide a high accuracy simulation of assemblies, the accuracy of the physics simulation depends on the accuracy of the models being used. While the computer-aided design (CAD) programs used for designing the models can accurately describe the geometric features of the models using non-uniform

rational basis splines (NURBS) or splines, the models need to be tessellated to be used in VR simulation. This leads to the loss of accuracy in the models, which results in parts that can be geometrically incompatible with each other.

Behandish *et al.* developed a generic and effective force model for virtual assembly [6]. Their approach utilised artificial energy fields around the virtual objects for collision detection between parts and to also guide the parts to their desired spatial configurations during virtual assembly A significant feat in their approach was its ability to unify the two phases of free motion and insertion into a single interaction mode effectively removing the need for different physics behaviour for each of the two phases. However, while their model was well suited for traditional peg-in-hole assembly, it does not provide an easy way of implementing more complex connections such as threaded connections.

Constraint-based modelling While PBM focuses on providing realistic physics-based interaction between parts, CBM sacrifices some of the physical accuracy for other benefits such as increased stability, better performance, accuracy, and simplicity. In the context of virtual assembly, CBM can be especially useful because the use of virtual constraints can be helpful for the user when physical constraints are missing. Systems using constraint-based modelling generally fall into two categories:

The first category uses positional constraints with pre-defined final part positions. In general, solutions involving this kind of constraints usually result in snap-to-place simulations where the parts are locked into their positions once the user moves them close to their target position. Since positional constraints are in practice trivial to implement, they are an attractive solution when modelling the interactions between parts is not crucial to the application. VR platform developed by Noghabaei *et al.* provided the possibility for identifying discrepancies in produced parts, and it can also work as a platform for interactive training and simulation [33]. While their approach was non-physical and parts connected to other parts with positional constraints, it showcases that VR assembly also works on large, industrial parts. One differentiating aspect of their research was the ability to import 3D scanned parts and examine their quality by trying to connect them to other parts.

The second category within CBM models the part-to-part connections based on the geometric features of the parts where geometric constraints are applied to restrict the relative motion of the parts when a certain criterion, usually distance between parts, is satisfied. The basic principle of these applications is to utilise a constraint solver that limits the way objects move relative to one another. Although multiple different approaches for solving systems of geometric constraints have been proposed, we do not go into details

regarding them as we are limited to using the constraint solver provided by PhysX, however, for more details of the methods used for constraint solving in VA, we suggest referring to a thorough review by Seth *et al.* [41].

Because of its benefits, CBM has been widely studied and utilised in previous research. Wang *et al.* discussed the basics of constrained motion simulation and provided methods and algorithms for checking and applying constraints in assembly simulation [50]. In their study, they investigated the analysis of combinations of axis and plane constraints and maintaining previously added constraints throughout the assembly process. They also suggested a method of guiding users in the assembly process by displaying constraints visually during the assembly process. Murray *et al.* developed a virtual environment for constraint-based assembly and maintenance task simulation and analysis of large-scale mechanical products [29] in their research regarding immersive assembly and maintenance simulation environment.

In addition to enforcing geometric constraints, some of the VA applications perform additional validation, such as collision detection, to ensure that the users' actions correspond to the real-life situation. The difference in the collision detection compared to the physics-based modelling is that in these VA applications the application merely checks that objects do not go through each other while in the physics-based applications the physics engine ensures that the objects cannot overlap. In their constraint-based assembly training system for an aircraft engine, Lu *et al.* used a hierarchical bounding box method for collision detection [27]. Although there was no physical response between the virtual parts, a force feedback equipment was integrated into the system to provide the user with proper feedback.

Hybrid modelling In addition to PBM and CBM, multiple hybrid approaches that try to combine the benefits of both methods have been proposed. Quite often these approaches combine physics and constraints by allowing physical interaction between virtual parts when they are not connected and disable collisions between two objects when a constraint between them is present thus allowing physical interaction between non-connected parts while maintaining the performance and accuracy benefits of CBM.

Tching *et al.* used a Virtuose haptic device to build an interactive simulation of CAD models assemblies [46]. In addition to using mechanical joints for limiting relative movements between two parts, they used virtual walls for guiding objects to a specific spatial configuration. They also formalised this concept as virtual constraint guidance (VCG) for insertion tasks. While their approach is closer to CBM than PBM, they allow physical contact between virtual parts until the insertion task reaches a specific state.

Similar way of combining PBM and CBM was also used by Seth *et al.* [40]. They utilised the B-Rep solid model data from the CAD model data for

both collision detection between virtual parts and for obtaining the geometric constraints. Their application used a custom physics engine for the simulation, and while this approach provides accurate simulation, we are not aware of any available physics engines that would support B-Rep based collision detection.

Gonzalez-Badillo *et al.* developed Haptic Assembly and Manufacturing System (HAMS), a physics and constraint-based haptic virtual assembly [19]. In their research, they found that users were able to perform the assembly tasks consistently faster when the constraints were enabled compared to when running the system without dynamic constraints. Furthermore, their users also perceived the controls to be easier to use when the constraints were active, which also translated to the assembly being easier.

2.2.2 Assembly assistance

To provide a meaningful VA training platform, it is necessary to provide appropriate assembly instructions as the user often has no prior knowledge about the task they are trying to learn. Although there exists plenty of previous research on generating step-by-step assembly instructions [3, 16], most of the studies focus on the offline generation of instructions and very few address the issue of providing real-time instructions based on the users' actions. Similarly, most of the applications utilising virtual assembly instructions have relied on a set of predefined instructions [39], and quite often these applications require the user to manually notify the application whenever the next instruction should be shown.

Recently, Khuong *et al.* have been one of the few to discuss context awareness and validation in assembly assistance [23]. However, since their application performed assembly guidance and error detection based on the occupancy of 3D voxel volumes, it is effectively limited to the use case where the user is assembling LEGO blocks. To provide a more generic model for managing context-aware assembly instructions, Claeys *et al.* proposed the usage of assembly instructions repository that would be used to provide instructions based on the various contextual information such as the information about the user and environment [14]. Although the use of instructions repository can be useful for adapting the instructions during run time, a large amount of different instructions would be required to provide context-aware step-by-step instructions that would allow the user to perform the assembly in an arbitrary order.

While context-aware assembly validation and assistance can be beneficial for VA, there is probably even more potential for it in augmented reality (AR) applications where the guidance could be utilised to aid users in their actual

work. Furthermore, recent developments in machine learning have made it possible to, for example, track users' workflow from first-person video [37], which combined with context-aware assembly guidance could allow real-time tracking and validation of users' work.

2.2.3 Game engines for virtual assembly

Like the term game engine suggests, game engines are mostly targeting game development. However, their extensive feature sets make them viable platforms for more generic applications as well since game engines usually support at least graphics, physics, audio, input and networking. For this reason, multiple studies have used a game engine as the underlying platform for building virtual reality applications related to virtual assembly.

Shiratuddin *et al.* used the Torque Game Engine (TGE) to develop a virtual resign review application for architectural designs [45]. Their study showcased how a game engine can provide all the needed capabilities for developing a system for reviewing virtual designs. Hu *et al.* proposed a new technique for developing virtual assembly applications using Unity 3D. Their system was capable of visualising assembly and disassembly steps for virtual assemblies [20]. Aziz *et al.* build their virtual mechanical assembly training application using Garry's Mod (GMod), a physics sandbox built on top of Source Engine [5]. In their application, the assembly was done using various menus rather than moving the parts to their correct places. Dwivedi *et al.* used Unity to develop a VA application where the user assembled a treadmill using Vive HMD and controllers [17]. To perform the assembly, the user had to move the part to a snap drop zone for the part to snap to its place. Unity was also used by Noghabaei *et al.* to create a simulation for pipe assembly [33]. In their study, they focused on using 3D scanned models of physical parts to detect manufacturing discrepancies and like Dwivedi *et al.* they used a snap-to-place approach for the assembly. Though these applications have been a step towards more immersive virtual assembly, they have not utilised the full potential of the game engines by neglecting the use of physics engine to provide more realistic interactions.

Unity is a cross-platform game engine that was released in 2005 and has since become one of the most widely used game engines [47]. For the requirements of the developed application, Unity provides multiple useful built-in features such as graphics engine, PhysX physics engine, and VR support. Besides, most of the VR hardware and Leap Motion also come with their SDKs for Unity integration. Although Unity mainly targets game developers, the abundant feature set makes it a viable choice for development of simulations, visualisations, and other applications.

While multiple other game engines such as Unreal Engine, with similar feature sets, are also available, we decided to use Unity (Long Term Support (LTS) Release 2018.4) purely based on our previous experience with it rather than due to any technical benefits. Oculus, Leap Motion, and SteamVR also come with pre-made Unity assets that can be used to get started with HMD and hand tracking quickly. Although Leap Motion also provides Leap Motion Interaction Engine, a system for Unity to interact with virtual objects using Leap Motion hands, we do not use it for interaction between virtual hands and virtual objects due to it not working as expected when trying to move objects connected. In particular, since the Leap Motion Interaction Engine does not use proper physics forces to move objects around, the joint solver fails to move connected parts together when one of the parts is grasped by a hand.

2.2.4 Hand-object interaction

3D interaction between user and virtual objects is an actively studied field in VR. While applications using wand-based user interfaces can rely on point-and-grab approaches, more natural user interaction can be achieved when the input interface does hand and finger tracking thus providing us with accurate skeletal data in the form of virtual hands. The interaction methods between virtual hands and virtual objects hereinafter referred to as hand-object interaction, can be divided into roughly two categories and combinations of these two.

Kinematic approaches A simple and commonly used method for hand-object interaction is to use kinematic grasps to pick up virtual objects. In these approaches, when a grasp is to be initialised, the virtual object is artificially connected to the hand so that hand movements are also applied to the grasped part. A standard method for kinematic grasping is to rely on predefined set of gestures, such as pinch gesture, to initiate the interaction [30, 31]. Leap Motion Interaction Engine also uses kinematic grasps to achieve simple interactions between hands and virtual objects. Due to their simplicity, kinematic gesture-based approaches are also appealing for machine learning as the gesture detection problem is well suited for supervised learning. Kinematic approaches are also well suited for VR applications due to their simplicity and low computational costs. However, one significant limitation in today's kinematic approaches is the lack of dexterous manipulation which significantly limits their usability in high accuracy tasks where the user is supposed to, for example, rotate objects with their fingertips. Having said that, while it would be possible to add dexterous manipulation to kinematic grasping, the usefulness of dexterous manipulation is likely to be limited by

the accuracy of finger tracking as well.

Physics-Based approaches One of the earliest physics-based approaches for realistic hand-object interaction was a spring-damper model proposed by-Borst *et al.* [9, 10]. In their approach, each of the virtual fingers tries to match the position of the corresponding physical fingers by using a set of imaginary springs. This essentially provides a one-way coupling between the physical hands and virtual hands where the state of the virtual hands can slightly differ from the state of the physical hands to avoid visual interpenetrations between hands and virtual objects. Similar approach was also used by Ott *et al.* with further improvements to two-handed haptic manipulation of objects [36]. Nasim *et al.* used a second dynamic proxy hand that gets enabled once a collision between the virtual hand and a virtual object is detected. Since in their approach the proxy hand gets frozen to the state in which the virtual hand was when the collision began, the interaction opportunities are limited to specific cases of grabbing and pushing [30]. Höll *et al.* proposed a friction-based approach for realistic hand object interaction where the penetration of tracked hands and virtual objects was used for computing and applying forces between virtual hands and objects [21]. While their approach makes dexterous manipulation of objects possible, it relied on phalanges of virtual fingers going inside the grasped objects which makes the approach unsuitable when dealing with small parts where virtual fingers do not penetrate deep enough to virtual objects. However, since in their approach grasping happens because of forces caused by fingertips and palms, the approach is more flexible in terms of holding multiple objects together with fingers. Although the possibility of dexterous manipulation makes physics-based approaches appealing for virtual assembly, the collisions between virtual hands and virtual objects may not be desirable as accidental hand movements could easily cause havoc in use cases such as virtual assembly where multiple objects interact with each other.

Hybrid approaches One way of combining the benefits of kinematic approaches and physics-based approaches is to allow collision between hands and virtual objects when the objects are not being grasped. Approach like this allows the user to use their virtual hands for both pushing the objects around and for picking them up. One such approach, suggested by Liu *et al.* is to use caging-based approach where the object becomes attached to hand when the geometry centre of collision points between the hand and the virtual object is inside the geometry of the virtual object [26].

While physics-based approaches can provide more realistic hand-object interaction than kinematic approaches, none of the existing approaches can robustly handle inaccuracies in the finger tracking which makes them unsuitable for virtual assembly of small objects with Leap Motion. Moreover, as

the physics-based approaches and hybrid approaches rely on physical contacts between virtual hands and virtual objects, we deemed them unsuitable for our application since it is easy to accidentally push virtual objects with hands and shake parts off from the assembly being built. For these reasons, we decided to use a simple kinematic approach where hands do not collide with virtual objects, and a pinch gesture is used to grab objects with hands.

Chapter 3

System design and environment

In this chapter, we describe the overall design of the application, including the hardware and software choices. First, in Section 3.1, we describe the toy set we use as a test case for the application. Then, in section 3.2, we list the requirements we set for the application ourselves. In Section 3.3, we present the overall system design and finally, in Section 3.4, we describe the hardware setup required by the application.

3.1 Test case

Although our goal is to develop a general application for virtual assembly, we use a set of Handy Man's Go-To Caddy toy blocks (Figure 3.1) as a test case for the application, and therefore some of the design decisions are based on the features of the toy blocks. The set consists of a total of 72 parts with 19 different varieties. While these parts are not an accurate representation of any real-life assembly tasks, they provide enough variety to demonstrate the flexibility of the application.

A notable aspect about the parts in the toy set is that nuts and bolts, the fasteners in the set are of similar size as the other parts in the set. Although the set does not contain tiny parts such as tiny screws that would require very accurate use of fingers, most of the earlier VA studies have been operating on much larger objects. Another notable difference compared to most of the existing VA applications is that the parts in the toy set are relatively small. While the largest part measures approximately 20 cm $\times$ 10 cm $\times$ 6 cm, the smallest grabbable volumes are merely 2 mm thick.

Figure 3.1: Parts that belong to Handy Man's Go-To Caddy

3.2 System requirements

To create a general virtual assembly training application that can easily be extended by introducing new assembly parts and assembly tasks, we set the following requirements to ourselves to act as guidance during the development:

1. Development is possible with a regular PC.
2. Application can easily be ported to different head-mounted displays.
3. User interface is not tied to any specific device. While we use Leap Motion during development, the hand-machine interface should only handle part movement so that it is possible to use other systems in the future.
4. Pieces for the assemblies are built in a modular way, making it possible to add new kinds of parts with low effort.
5. The application should feel convincing and realistic within hardware capabilities. While realism is a difficult value to measure, we believe that if the user can become immersed in the application, the application does satisfy the requirement.
6. Application should support loading different assembly tasks. Assembly tasks, including virtual parts used for the assembly, are provided in this project as Unity assets with possible future support for loading them directly from file system as well.

7. Application should also be able to take into account possible inaccuracies in the assembly task data. Pham *et al.* used unsupervised workflow extraction to extract assembly tasks from first-person video [37]. To support similar assembly extraction methods, our application should be able to resolve connections based on relative positions and orientations of parts.

8. Application should support instructing user through the assembly process and detect mistakes done by the user.

3.3 System design

In this section, we describe our overall system design in three parts. First, we describe how we represent assembly and part data in our application. Then, we go through the constraint-based modelling approach we adapted from earlier research, and finally we present the modular structure of the application.

3.3.1 Assembly data

Multiple ways of representing assembly data have been suggested in the past, and it seems to be widely accepted that humans understand assembly as the process of repeatedly connecting parts to other parts. Therefore it is no surprise that one of the widely used approaches for representing assemblies has been to treat them as collections of parts that contain various features that can connect to each other (Figure 3.2). Based on this convention, we use the following breakdown to represent the virtual assemblies:

- **Assembly** is a collection of interconnected parts, and single part also forms assembly by itself.

- **Part** is the smallest physical piece that can be interacted with by the user. Parts contain attachments, features that can be connected to other attachments.

- **Attachment** is a single feature in part that can form connections with attachments in other parts. See Figure 4.3 for an example of part-attachment composition.

Although some applications, such as [3], have treated fasteners as special cases, we handle them as regular parts since in our test case fasteners are no different from other parts (Section 3.1). For a real-world scenario where

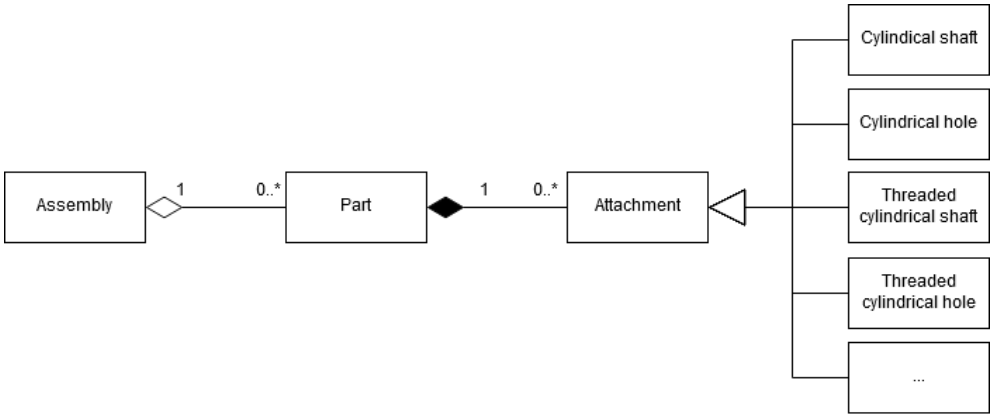


Figure 3.2: Relationships between assemblies, parts, and attachments.

fasteners would be much smaller compared to other parts, we may want to revisit this decision but treating fasteners as parts has the added benefit of not having to handle some parts differently. Moreover, it is not uncommon for mechanical parts to have threading integrated into the part itself, which is another reason we would not suggest handling fasteners separately.

3.3.2 Constraint-based modelling

For virtual assembly, we use a combination of PBM and CBM similarly to Tching *et al.* [46]. In their application, the assembly happens using physics constraints, but when two parts are not connected, they can collide with each other with proper physical feedback. We believe that this approach is well suited for implementation running on top of a game engine as physics engines used in game engines can handle both collisions and constraints between virtual parts. Furthermore, since most of these physics engines have been made with game physics in mind, they can handle numerous physics objects and constraints in real-time which is especially crucial for VR applications which require high refresh rates to provide a smooth experience and to avoid motion sickness.

Another reason for choosing a combination of PBM and CBM is that PBM alone tends to have higher computational costs since it relies on accurate contacts between physical parts. Moreover, the use of physics constraints for the assembly provides an easy and computationally cheap way to aid the user in connecting parts together which is also essential considering that the accuracy of the available hand tracking solutions is still somewhat limited. This has been found to be beneficial for the assembly process compared to using plain physics-based assembly.

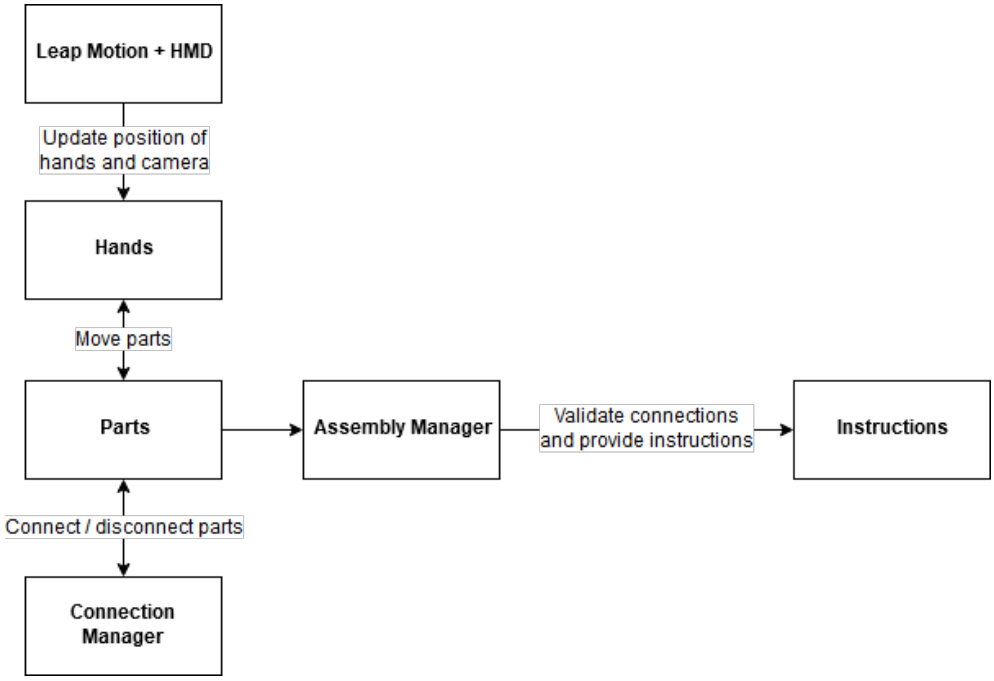


Figure 3.3: System overview and interaction between different modules.

3.3.3 Application structure

Our application is structured in a modular way where all of the active systems work as standalone systems and thus have no direct interaction between each other (Figure 3.3). That means that, for example, the part of the application that handles the constraint-based assembly has no knowledge of how movement of parts happens with the use of virtual hands and therefore it is possible to replace the virtual hands with another implementation without affecting any of the other systems. With this in mind, our application consists of the following active systems:

- **Hands** are moved in VR according to users' physical hand movements and can be used to grab parts inside the simulation and move them around.
- **Connection Manager** creates guides and connections between attachments when their relative alignment is favourable. It also takes care of merging and splitting assemblies whenever parts are connected to, or disconnected from each other.
- **Assembly Manager** handles assembly validation and provides real-time visual assembly instructions for the user based on the connections

Figure 3.4: The hardware setup used in the application.

between parts. Additionally, the assembly manager is also in charge of loading target assemblies and initialising the parts for the user.

- **Speech Manager** listens for voice commands and plays spoken instructions whenever needed.

The data required by the application, in particular assemblies and the parts used in the assemblies, are provided as prefabricated Unity objects, prefabs, that are built using Unity editor and loaded during the run time as this allows an easy way for us to author the data. While this sets the requirement of using Unity editor for adding new data, adding support for loading the part and assembly definitions from files can be done in the future if need be.

3.4 Hardware

Our hardware setup consists of a head-mounted display and hand machine interface as pictured in Figure 3.4. In this section, we provide information on the reasoning behind our hardware selections.

3.4.1 Head-mounted display

One of the goals of this thesis is to create an application that can run on different kinds of hardware. To satisfy this requirement, we use two different HMDs, Oculus Rift [35] and HTC VIVE [49] for development. While the underlying hardware specifications of these 2 devices are identical in terms of

display resolution, refresh rate, and field of view, the headsets differ in both their tracking methods and the underlying software implementations.

It is also worth mentioning that since HTC VIVE uses SteamVR, the application is expected to work out-of-the-box with other SteamVR-compatible devices, for example, Windows Mixed Reality headsets, as well. With this in mind, we expect the application to run on most of the available consumer VR devices, excluding mobile VR.

3.4.2 Hand machine interface

To achieve natural user interaction between the user and virtual parts, we let the user interact with virtual parts using virtual hands. To do this, we use Leap Motion [24], a device that provides accurate hand and finger tracking using infrared cameras and infrared LEDs. While we initially considered using CaptoGlove [13], a wearable hand machine interface for gaming and smart devices, we opted to use Leap Motion because our application required positional tracking for hands which was not available for CaptoGlove without purchasing an additional sensor that was not available when we were deciding which user interface to use. Positional tracking was, however, supported by Leap Motion when using it together with a VR headset.

Since leap Motion is both affordable and reliable, it has been widely used in previous studies when hand-object interaction has been required. Since we were aware of possible problems with tracking accuracy and narrow tracking cone, we also considered the possibility of using controllers of the VR headsets as a backup. Although Leap Motion officially supports Oculus rift and VIVE, it is known to work on other PC compatible VR headsets as well. Moreover, upcoming Leap Motion Mobile Platform will provide seamless integration to standalone VR headsets such as Gear VR so in future it would be possible to make an untethered version of the application.

Chapter 4

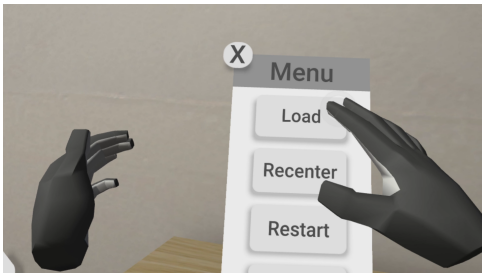
Implementation

This chapter describes the system that was developed. Section 4.1 describes the basic usage of the application. In Section 4.2, we go through the methods for obtaining the 3D objects for parts and the process of annotating them with information necessary for virtual assembly and in Section 4.3 we describe how these part definitions are used to create assembly tasks. After that, in Section 4.4, we discuss in detail how the constraint-based assembly is handled in our application. Then, in Section 4.5, we discuss the method we use for interaction between virtual hand and virtual objects. Section 4.6 contains information about the scene setup we used for the user test. In Section 4.7, we describe our method for assembly validation which is followed by a Section 4.8 describing assembly assistance. Lastly, in Section 4.9, we discuss some of the unaccounted physics behaviour observed during the development.

4.1 Usage scenario

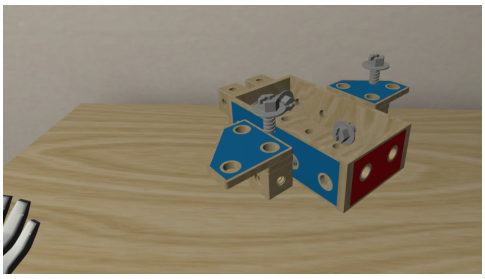
The basic workflow in our application is as follows: When the program is started, the user needs to find a spot within the field of view of the VR tracking stations where there is enough room for them to stand and to move their hands around. To minimise possible interference with Leap Motion, the user should not be pointing towards reflective surfaces, and it is also recommended by Leap Motion guidelines to remove wristwatch and other reflective objects from hands.

To interact with the application, the user may use either voice commands or a touch menu (Figure 4.1b) that can be opened by tapping a button attached to users' wrist. The available actions user can perform depend on the state of the program, but during the basic workflow user will have to utilise at least the following following actions: *start* to start the application,

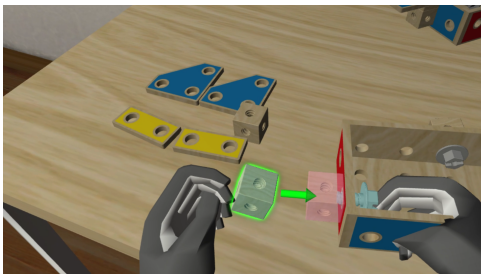


(a) User is navigating the touch menu.

(b) User lists available voice commands.



(c) Exploded view of the assembly.



(d) User is following assembly instructions.

Figure 4.1: Different stages of the users' workflow.

list to list available assembly tasks, *load* to load an assembly task, and *quit* to exit the application. Additionally, the user is provided with a *voice*-command to list all available voice commands (Figure 4.1b).

If the application was started before the user got their headset on, they can recenter the headset tracking and set the viewport to the correct starting position. Once ready to begin, the user needs to start the simulation by initiating the *start*-action. After this, the user will get a short, spoken in-application introduction about the available commands. After the introduction, the user is expected to load the actual assembly task by first listing the available assemblies and then loading the only available assembly called *Medium*. Once the task has been loaded, the user is presented with an introductory animation where we show the user the full assembly they are about to build and also an animation depicting the exploded view of the assembly to showcase the part-to-part connections (Figure 4.1c). After this, we move the assembly to the table in front of the user so that the user may inspect it later if need be. Once the animation finishes, we also spawn the parts user needs to assemble to the table so the user can begin assembling the parts.

To instruct the user in the assembly process, we present the user with visual instructions on what parts should be joined together next (Figure 4.1d), or if there is an error and user should disconnect part from another

Figure 4.2: Bolt medium hierarchy

instead (Section 4.8). These instructions are automatically updated every time the user connects or disconnects parts. The user moves the virtual parts or assemblies around by grabbing them with their virtual hands and moving the hands around (Section 4.5). Once two parts are close to each other, a virtual constraint, a guide, is activated between them to restrict their relative movement to aid the user in connecting the two parts (Section 4.4). Once the guide-specific connection criteria are satisfied, the guide is changed into connection and the two assemblies the parts belonged to are merged to a single assembly. Once the user considers the assembly to be ready, they can exit the application with the *quit*-action.

4.2 Part definitions

We use Shapr3D [42] to create 3D CAD models of the toy blocks. Although any 3D modelling software would suffice for the task, the use of CAD software has the added benefit of being closer to the real-life case where parts are designed in CAD before production. To import the objects to Unity, we export the CAD models as STL files and convert them to FBX models using Blender [7]. Although Shapr3D can export Unity-compatible .obj files directly, we need to process the models in Blender to assign correct materials to the geometry and to perform UV-mapping to set up textures. Since most of the VR headsets need to render the scene twice, once for each eye, we also take the opportunity to apply a decimate filter to the models in Blender to keep the number of polygons as small as possible while still maintaining the visual integrity of the models.

To support virtual assembly, the 3D models of the parts need to be augmented with geometrical and mechanical information describing the different attachment points of the part. We do this by building Unity prefabs, prefabricated objects that act as templates for run time object creation. Each of our virtual part prefabs consists of a root object containing a *Part*-component that contains the basic information about the part such as name and mass.

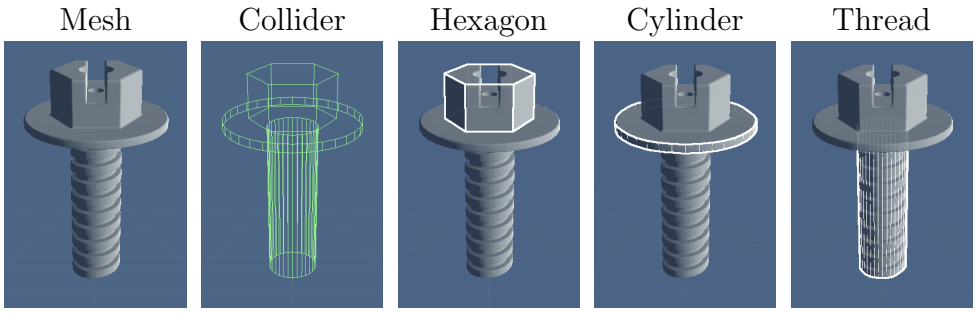


Figure 4.3: Composition of bolt medium. In addition to the visual model, the part consists of 3 attachments: Polyhedral shaft as head, cylindrical shaft as washer and threaded cylindrical shaft as threads which together form the collider of the part.

Also, each of the parts contains one or more attachments provided as child objects (Figure 4.2).

As described in Section 3.3, our application handles connections between parts as attachment to attachment connections. We implement attachments as components that inherit from *Attachment* base class (Figure 3.2). Different attachment types are used to determine what kinds of attachments are capable of forming connections with each other as well as to describe the shapes of the attachments. We implemented a total of 7 different attachments for our application: box, cylindrical hole, cylindrical shaft, line hole, polyhedral hole, polyhedral shaft, threaded cylindrical hole, and threaded cylindrical shaft.

To detect collisions between moving objects, Unity requires the colliders of those object to either be primitive colliders (box, sphere, or capsule) or convex mesh colliders. Furthermore, the number of faces and vertices in convex mesh colliders is limited to 255 which significantly limits the complexity of collider shapes. However, when more complex colliders are required for moving objects it is possible to create a compound collider out of these simple shapes. Considering that most of our parts are concave, we are also taking this approach to provide accurate collider shapes for the objects. Conveniently, since each of the attachments is aware of its shape, we use the attachments to construct compound colliders for the parts (Figure 4.3).

Although it would be possible to deal with holes in the parts with the use of compound colliders, it would not be feasible in practice as it would come with a high performance impact making the approach unsuitable for VR. However, with constraint-based assembly, we can emulate the holes by disabling collisions between the object the hole belongs to and the object

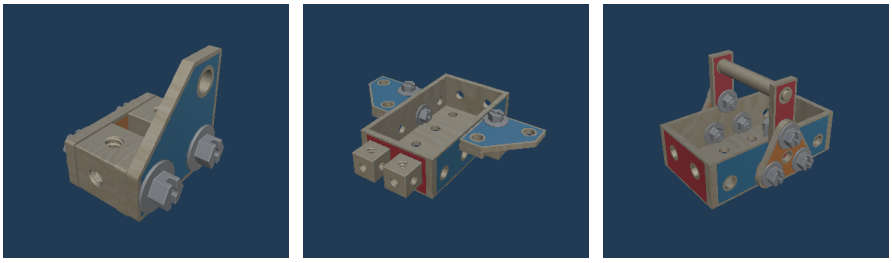


Figure 4.4: Easy, medium, and hard assembly tasks included in the application.

entering the hole when the movement of the two objects is restricted so that they can only move along the direction of the hole. This approach used by Tching *et al.* [46] is well suited for real-time VR applications since it does not require additional computation apart from the physics constraints that are already present and it may, in fact, actually reduce the strain from the physics engine as there are fewer collision checks to perform. One small drawback in this approach is, however, that objects cannot enter holes when they are not actively under a constraint but this issue can be mostly mitigated with careful design of constraints.

4.3 Assembly definitions

In addition to providing the virtual parts as Unity prefabs, we also provide the target assemblies as Unity prefabs as this provides a convenient way of authoring new assembly tasks inside of Unity editor. We include 3 different types of assembly tasks: easy, medium, and hard, in our application to test different levels of complexity (Figure 4.4).

When loading assembly tasks, we automatically deduce the connections between each of the parts instead of relying on connection info specified beforehand. The reason for this is that this kind of approach allows for working with assembly data extracted from, for example, first person videos [37] where it is often possible to obtain relative positions and rotations of two parts but where the information of how the parts are connected is not available.

To resolve the connections between parts, we perform a naive loop through all pairs of attachments and check whether they satisfy certain criteria related to each of the attachment types required for the connection. As an example, in the case of a cylindrical shaft and a cylindrical hole, we calculate if the shaft penetrates the hole.

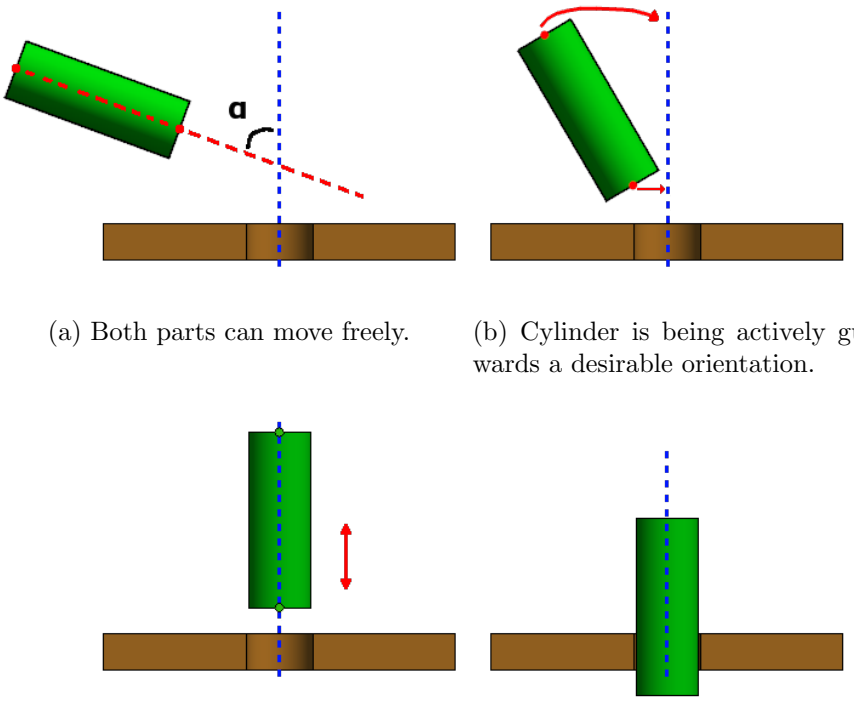
4.4 Constraint-based assembly

We base our implementation of constraint-based assembly mostly on the concept of virtual constraint guidance (VCG) by Tching *et al.* [46]. The main idea of VCG is to combine virtual constraints that guide the object to a specific spatial configuration with mechanical joints that limit the relative movement of two parts. Although Tching *et al.* focused in the traditional peg-in-hole assembly, the underlying idea of their implementation is general enough to lend itself to different kinds of mechanical linkages such as screw connections or even more complex connections.

Unlike Tching *et al.* who used virtual walls for guiding the peg into the hole and kinematic constraints for limiting the relative movement between connected parts, we use kinematic constraints not only for forming the connections between parts but also for guiding the parts to a favourable alignment required by the connection. The reason for this change is that having the virtual walls attached to the object with a hole would not be practical when both of the parts can be manipulated by the user. Moreover, the use of kinematic constraints for the guidance gives us more customizability on how different connections behave.

To activate the guidance in our simulation, the user needs to place two parts in a spatial configuration where two compatible attachments are within a certain distance of each other, and their relative orientation is within certain angular threshold. Finding suitable values for the thresholds depends on the size of the parts and desired level of guidance, but with empirical tests, we came up with a distance threshold of 0.35 world units (effectively 35 cm) and angle threshold of 45° for our connections (Figure 4.5a). When a guide is to be activated, we introduce a kinematic constraint by creating a *Configurable Joint* that limits the relative movement of the parts. In Unity, *Configurable Joint* is a physics joint that allows the highest level of customisation among all the possible joints is therefore best suited for our use case.

We treat guidance and connecting as a two-stage process. In the first stage, the spatial positioning of the parts does not allow forming the connection, for example, due to collisions between parts. In this stage, the physics joint is configured so, that by using spring forces, the parts are actively driven towards the spatial configuration required by the second stage. In the second stage, the parts are aligned so that the connection between parts can be formed by, for example, sliding a cylindrical shaft into a cylindrical hole. Once the alignment between two parts satisfied the required spatial configuration, the guide moves to the second stage where the relative alignment of the parts is locked, and the relative movement between the parts is re-



(a) Both parts can move freely.

(b) Cylinder is being actively guided towards a desirable orientation.

(c) Relative movement is restricted along the primary axis of the hole.

(d) Parts are connected to each other.

Figure 4.5: In Figure 4.5a the movement of the cylinder hasn't been constrained. In Figure 4.5b the movement of the cylinder is being guided towards an spatial configuration that allows the cylinder to slide freely into the hole. In Figure 4.5c the rotation of the cylinder has been locked and the relative movement between the parts is restricted only along its primary axis towards the hole. In Figure 4.5d the cylinder has been slid into the hole and connection between the two parts has been formed.

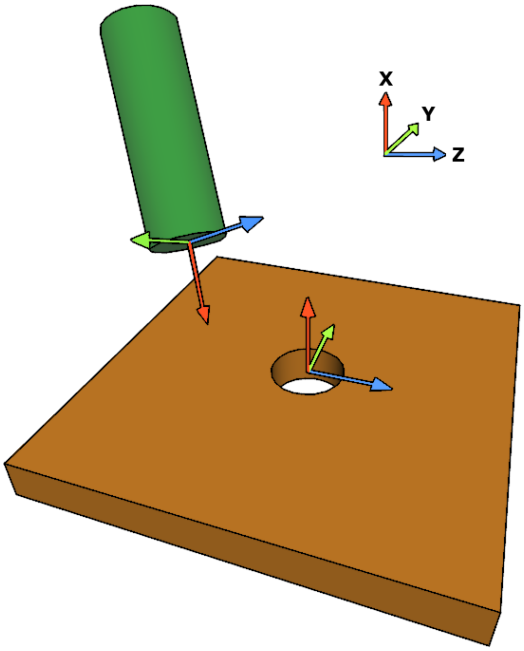


Figure 4.6: Axes of the configurable joint

stricted to the axes where the parts need to approach each other to form the connection.

When entering the second stage, we also disable some of the collisions between the two parts if there are attachments that need to be able to penetrate each other as in the case of cylindrical shaft entering a cylindrical hole. To do this, we find the collider the hole belongs to and disable the collision between that collider and the collider of the shaft. Unlike Tching *et al.* who disable the collisions between the two parts, by selectively disabled collisions between specific attachments instead of disabling the collisions between the two parts, it is still possible, for example, for the head of a bolt to collide with the object being penetrated by the threads of the same bolt.

Some special consideration is required when initialising the *Configurable Joint*. First, the joint axes need to be constructed in a way that the sliding motion for forming the connection happens on joints' X-axes because some of the angular joint parameters, such as *Angular X Drive* and *Angular YZDrive*, couple configuration of angular Y and Z motion together while still allowing the configuration of X motion separately (Figure 4.6). Besides, when creating a *Joint* in Unity during runtime, it is necessary to align the objects to be jointed in the target configuration when creating the joint.

Once a connection reaches a specific state, such as a shaft enters a hole, we start considering the two parts to be connected, and at this point, we also

merge the parts' assemblies to a single assembly. Likewise, when two parts are disconnected from each other, for example, due to a shaft exiting a hole, we check the connectivity of the parts in the assembly using a breadth-first search and split the assembly into multiple different sub-assemblies. While the concept of *Assembly* is not strictly required by any of the methods used in the application, we do it for mostly practical reasons as it provides us with an easy and fast way of tracking sets of interconnected parts.

4.4.1 Guides and connections

To meet the design goal of creating an application where introducing new parts is easy, the logic for determining if an attachment can be connected to another attachment is provided by one of the two attachments. By having this high-level logic coupled together with attachments, it is possible to easily introduce new attachments capable of connecting to existing attachments without having to modify the old attachment. Moreover, to facilitate reusability of existing guides and connections, the logic related to guides and connections is separate from the attachments, and therefore attachments merely specify what kind of guides and connections can be used to join the attachment to another attachment.

For our test, we provide two kinds of connections, a shaft to hole connection and a threaded shaft to threaded hole connection. While the parts in the toy set could support three more connection types, polyhedral shaft to polyhedral hole, box shaft to box hole, and shaft to line hole, due to time constraints we chose to implement the aforementioned connections for our user testing to ensure the capability of introducing different connections, and to get a feeling how the screw connection works without support for dexterous manipulation of objects with fingers.

Shaft to hole connection is a traditional peg-in-hole connection formed between either cylindrical shaft and cylindrical hole or threaded cylindrical shaft and cylindrical hole (Figure 4.5. In shaft to hole connection, the shaft can move freely in the hole along its primary axis. This functionality is achieved by setting the Configurable Joint's *xMotion* to limited and locking the *yMotion*, *zMotion*, *angularYMotion*, and *angularZMotion*. In case of shafts with a cap, limiting the *xMotion* to length of the shaft is also used to aid physics solver in cases where the part shaft belongs to could go through the hole due to physics running in discrete time steps.

Threaded shaft to threaded hole connection is a screw connection formed between threaded cylindrical shaft and threaded cylindrical hole. To form the connection, the following criteria must be met: First, the diameter of the threaded shaft needs to match the diameter of the threaded hole. Sec-

ond, the handedness of the threads need to be the same in both attachments. Finally, the size of the threads needs to be the same in both. While all of the threads in our test objects are of the same size, diameter, and handedness, it may be desirable to also allow smaller threaded shaft to larger threaded holes using a regular shaft to hole connection if needed. A significant implementation difference in threaded shaft to threaded hole connection compared to its threadless counterpart is that the *xMotion* of the joint is set to locked, and we actively track the relative rotation between the shaft and the hole, and update the joint *anchor* based on the accumulated rotation to have the shaft screw inwards to the hole or screw outwards from the hole based on direction of rotation thus providing a scew-like behaviour.

4.5 Hand-object interaction

While experimenting with different ways of implementing hand-object interaction we ran into several problems. Because Leap Motion only sees your hands when you see them, it is easy to accidentally have Leap Motion lose track of your hand leading to a situation where the hand is teleported from one place to another when it becomes visible again. This naturally adds some unwanted side effects to the simulation as hand may, for example, be placed inside another physical object without having a chance to trigger a proper physics response. Also, due to lack of physical force feedback when operating with Leap Motion, it is natural and common for user to place hand inside solid objects or to pinch fingers through grasped objects and therefore, after experimenting with different hand-object interaction methods we came to a conclusion that having collisions between virtual hands and virtual objects had more negatives than positives when trying to assemble a mechanical contraption. Naturally, this decision limits us from performing certain actions such as grasping a stack of parts between fingers (Figure 4.7) or holding a bolt against another part with fingers while connecting a nut to the bolt with another hand.

Oculus Developer blog had suggested the following possible ways of grasping to objects: teleporting object to correct position, using physics joints to connect the object to the hand, attaching object to the hand directly, and keeping the object at the correct position in relation to the hand by each frame applying force and torque needed to force the object to the desired position [34]. While the first option was out of question due to the fact that physics joints connecting objects to each other do not play well with teleporting one of the jointed objects, the second alternative had issues as well due to the fact that that PhysX joints do not behave as expected when ob-

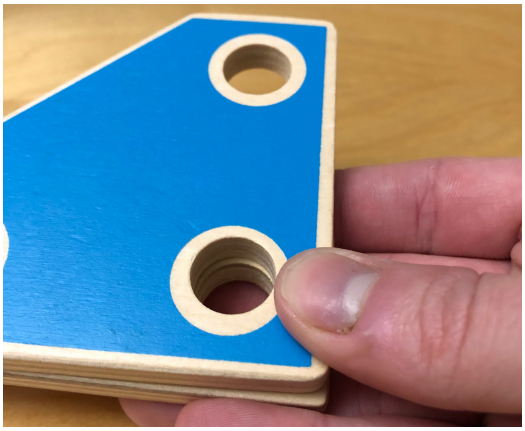


Figure 4.7: Example of a grasp that is not possible to achieve with hand-object interaction models that handle the grasping of objects in per-object basis rather than handling the interaction as forces between hands and objects.

jects are jointed to kinematic rigid bodies, the hands. This same behaviour was also present when the object was attached to the hand directly through parenting thus forcing the grasped object to be treated as a kinematic rigid body as well. Although the fourth and last method gave us promising results in the beginning, it tended to be slightly too fast when following rapid hand movements which often led to attached objects being forced to impossible positions.

When testing multiple different approaches for hand-object interaction, we learned that to use constraint-based guidance, we need to allow the part to move slightly between the virtual fingers when it is being actively constrained by a virtual constraint. A real-life correspondence for this behaviour would be skin of the fingers deforming and joints of the fingers adjusting to the movement of the object being held. For this reason, we refined the idea from transform matching as suggested in Oculus Developer blog, but instead of applying the computed force directly, we used a spring-damper model based on [32] to calculate the applied force and torque to the object.

To detect if user wishes to grab an object with a hand, we use an approach similar to Nasim *et al.* where the grab is initiated based on if the hand is close to being a fist [31]. In their approach Nasim *et al.* calculated a grab strength GS where $[0 \leq GS \leq 1]$, based on the distance between all fingers and the thumb with $GS = 0$ indicating an open hand and $GS = 1$ indicating a closed hand. Then, if there is a grabbable object close enough to the palm of the hand and GS is sufficient enough, the object is moved closer to the

hand and considered grasped. In our approach, however, because of the small size of the objects, we found the application to be easier to control if the GS is only computed based on the distance from the thumb to index finger and middle finger. Additionally, instead of looking for object closest to the palm we grasp the object closest to the imaginary centre between the 3 fingertips. To detect the grasp intent we use a fixed threshold of $GS \geq 0.8$ and similarly, the object is released if $GS \leq 0.75$. The purpose of the 0.05 difference between grasping and releasing is to work around inaccuracies in the finger tracking to avoid accidentally releasing the object when GS is just barely above the threshold.

When an object becomes grasped by hand, we store the position and rotation of the grasped object relative to a reference point in the grasping hand. We also disable the drag, angular drag, and angular velocity limit of the *RigidBody* component of the grasped object since the movement of virtual hands is directly connected to the movement of physical hands, and thus the hand movement is unaffected from drag and other forces coming from the physics engine. Then, every physics update running at fixed timestep Δt, we compute the position and rotation of the grasped object as if it were directly attached to the grasping hand. By comparing these values to the actual position and rotation of the part, we obtain positional and angular displacement, x and θ, correspondingly. The behaviour of the spring is controlled with two coefficients, spring coefficient C_k and damping coefficient C_d where $[0 \leq C_k, C_d \leq 1]$. Based on empirical testing, we settled with $C_k = 1.0$ and $C_d = 0.75$.

For an object with mass m moving, velocity v, and moment of inertia I, we use the following formulae for force f and torque τ for positional and rotational springs, correspondingly:

$$f = -\frac{m}{\Delta t^2} C_k x - \frac{m}{\Delta t} C_d v \tag{4.1}$$

$$\tau = -\frac{I}{\Delta t^2} C_k \theta - \frac{m}{\Delta t} C_d \omega \tag{4.2}$$

One attractive property in this approach, unlike transform matching or parenting, is that the spring and damper coefficients can be adjusted so that in case of rapid hand movements the grasped object experiences smoother movement towards the target position rather than being instantly teleported there which helps in reducing accidentally shaking connected parts to disconnect from each other. This does, however, produce a rather stiff connection between the hand and the grasped object, so, to avoid extreme speeds when hands get teleported due to occlusion, we cap the force and torque to maximum magnitudes of 25.0 and 0.75, correspondingly.

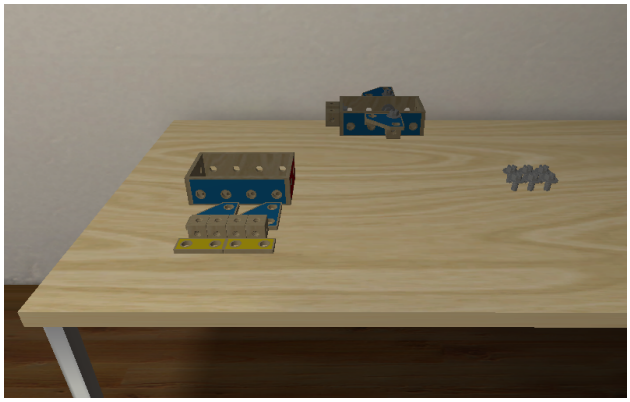


Figure 4.8: Scene setup where user stands in front of the table and operates the parts with their hands.

4.6 Scene setup

During the development, we noticed that the way the working area in the scene is organised makes a significant impact on the usability of the application. More specifically, since Leap Motions loses tracking of the hands whenever they go out of the viewport of the Leap Motion cameras, we concluded that setting up the scene in a way that tries to minimise the amount of head turning required helps the user in avoiding accidental loss of hand tracking. Leap Motion also suffers from poor tracking accuracy when the user tries to reach far away as many of the fingers get occluded while doing so.

However, minimising the amount of head turning required is a challenging task considering that the horizontal FOV of Oculus Rift and HTC Vive is only 110° compared to the 210° FOV of the human vision. To facilitate this, in our scene setup (Figure 4.8) the user stands in front of a table so that the parts used for the assembly are divided to two groups on the left and right of the user and are well within users reach. We also place the target assembly in front of the user but move it further down the table as it is intentioned to be more of a supplementary information in addition to the visual guidance. User may, however, grab the target assembly to their hand to inspect if needed.

4.7 Assembly validation

Although there has been wide research on virtual assembly and AR assisted assembly, only a few address the topic of assembly validation. As one of our

design requirements is not to restrict the user to a specific way of constructing the assembly, we cannot assume that the user performs the assembly following a specific fixed-order instructions. Therefore, we need to perform run-time assembly validation every time the user connects or disconnects parts. While there have been some previous attempts such as [23] for run time assembly validation, we could not find any that would be suitable for our use case. Therefore, we implemented a custom method for assembly validation that works by matching assemblies built by the user to the assembly user is supposed to build.

Since many of the parts in the toy set are symmetric, to perform the assembly matching, we need a way of first detecting the part symmetry and then to take the symmetry into account. Therefore, for each of the parts, we precompute attachment-to-attachment mappings for all possible rotations of the part that would keep the part symmetrical in geometry's perspective. That is, if the part is rotated, all the space occupied by the part before rotation should be occupied by the same kind of solid attachments after the rotation. Likewise, all the holes should match to a similar hole occupying the same space before and after the rotation. To check if the space occupied by an attachment matches the space occupied by another attachment when the part is rotated, we compute the intersection volume of the convex collider of the parts, and if the volume is within a certain threshold, we consider the attachments to match each other.

Because all of the parts used were symmetrical in rotations multiple of 30 degrees, we implemented a brute force approach for symmetry computation where we check for symmetries by testing all possible rotations where Euler angles were multiples of 30. This leads to a set of 1728 rotations that need to be tested, but since some of these rotations result in same orientations we end up with having to test less than 1000 rotations, which, due to efficient matching of convex polyhedrons, happens in less than a second even for the most complicated of our parts. Although this leaves some room for optimisation, we do not consider it necessary as the computation can be done offline outside of the program execution.

To validate the assembly, we use a custom procedure that finds multiple one-to-one mappings from parts operated by the user to the target assembly they are trying to build. From here on, we refer to assembly user is trying to build as *target assembly* and the parts in it as *target parts*. Likewise, we refer to the assemblies operated by user as *working assemblies* and parts inside them as *working parts*. Moreover, we call the one-to-one mappings between *working parts* and *target parts* as *mappings*.

Our algorithm works by going through all the working parts one by one, and finding correspondences for them in the target assembly based on the

current one-to-one mappings between working parts and target parts and updating the mappings on the go. Because of symmetries, when adding a part to a mapping, it is possible to find multiple correspondences as the introduced part can be rotated around. Therefore, each added part may introduce multiple new one-to-one mappings instead of just a single one. Similarly, it is possible that no corresponding parts can be found in which case, the one-to-one mapping can be treated as invalid, and therefore it can be discarded.

Let $M_i = \{w_j \rightarrow (t_k, s_l)\}$ be a set of one-to-one mappings from working part w_j to a target part t_k rotated by a rotation s_l. Also, let $R_n = \{M_i\}$ be a set of these mappings at the beginning of matching nth part in the working assemblies.

Initially, we start our search by having a single one-to-one mapping, $M_0 = \{\}$, $R_0 = \{M_0\}$ where none of the working parts have been mapped to target parts. Then, we start going through working assemblies one by one and trying to match the working assemblies to the target assembly. When processing a working assembly, we start by selecting the first part of the assembly as a root part w_0, a part we use as a starting point when searching for matches. Although the algorithm provides the same results regardless of how root parts get selected, there may be small performance gains available by selecting root parts that provide fewer matches in the target assembly as the algorithm would have fewer combinations to check when advancing further when processing the working assembly.

Once we have selected a root part from a working assembly, we go through all the mappings $M_i \in R_0$ we have obtained so far, and with each M_i we look for target parts, that are of the same type as the root part and that have not been mapped to any of working parts yet. If a non-mapped part of same type is found, we go through all symmetries s_l of the root part, and for each of the symmetries we create a new mapping that contains all one-to-one mappings in M_i and the mapping from root part w_0 to (t_k, s_l) or more formally $M_{i*} = M_i \cup \{w0 \rightarrow (t_k, s_l)\}$ and $R_{n+1} = R_{n+1} \cup M_{i*}$

Similarly to a flood-fill algorithm, starting from the root part w_0, we start looking for neighbouring parts w_j based on the part-to-part connections. Going through all the mappings in R_n, we have two separate cases to handle. If the part w_j has already been visited but not through the connection being processed, we need the check connections from the corresponding target part t_0 and discard the mapping if there is no corresponding connection to be found from t_0 to t_j.

However, if the part w_j has not been visited yet, we look for a connection from matched target part t_0 to another part t_j in the target assembly where the part t_j has not been mapped to any of the working parts yet. Then,

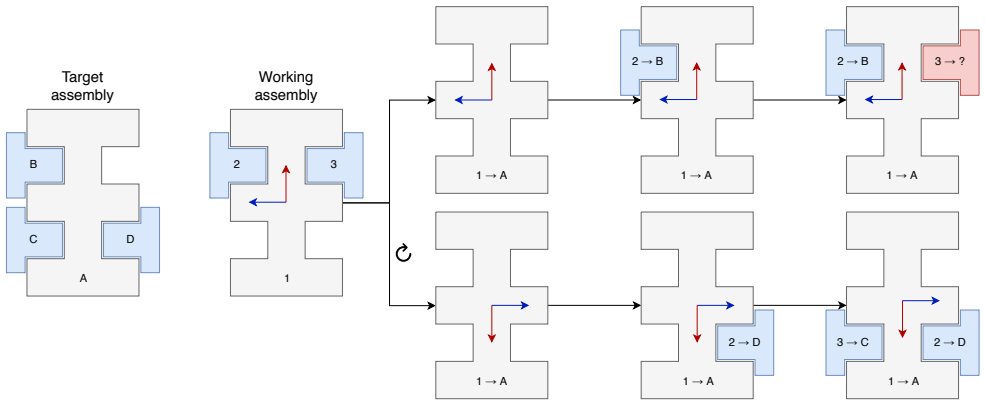


Figure 4.9: Example of matching procedure where one match between working assembly and target assembly is found by mapping $1 \rightarrow (A, 180°)$, $2 \rightarrow (D, 0°)$, and $3 \rightarrow (C, 0°)$. Upper branch of matching fails to find a correspondence for part 3 and is therefore discarded.

once again, going through all symmetries s_l of the working part w_j, we try to find all symmetries where the connection between t_0 and t_j matches the connection between w_0 and w_j and if such a connection is found, we create new a mapping that contains all one-to-one mappings in M_x and the mapping from w_i to (t_i, s_j) and add the mapping to R_{n+1}.

Once all of the parts in the working assembly have been processed, we proceed to match the next working assembly until all the working assemblies have been processed. After all of the working assemblies have been matched to the target assembly, we perform additional validation step where we prune all mappings that conflict with required precedence relations in the assembly. That is, we discard the match if there is a connection c_i that needs to be satisfied before connection c_j but that is not satisfied (Figure 4.10). However, a slightly more sophisticated solution would be required if there were parts that could go entirely through the hole as this simple approach is limited to non-conditional precedence relationships and this simple approach was chosen due to it being sufficient for the toy set being used. Though it would be possible to check these requirements already when matching the parts and connections, we found it more straightforward (although potentially slightly slower) to do it after all the mappings have been obtained.

Finally, once we have obtained and pruned all of the one-to-one mappings, we go through all of them and map the parts in working assemblies containing only a single part to arbitrarily chosen non-mapped parts of the same type in the target assembly. Although this is not strictly necessary, we do it for the convenience of not having to handle those parts separately from other

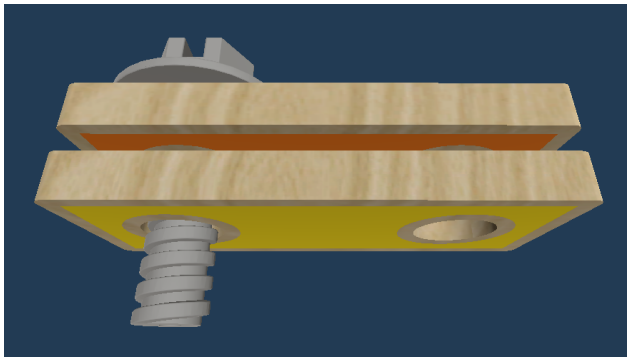


Figure 4.10: Precedence requirement where the orange part needs to be connected to the grey bolt before the yellow part.

assemblies when using the one-to-one mappings for assembly assistance. Similarly, if all the working assemblies are single part assemblies, the algorithm produces a single one-to-one mapping where parts are arbitrarily matched to corresponding parts.

4.8 Assembly assistance

To give the user the freedom to perform the assembly the way they feel convenient, instead of having fixed instructions on how to perform the assembly, we provide context-aware instructions to users. To do this, each time the user connects or disconnects parts, we check the validity of the assembly (Section 4.7) and show instructions based on that (Figure 4.11). These instructions involve either connecting two assemblies or disconnecting a part from an assembly.

If assembly is in an invalid state, we highlight the last added part with red colour to signal the user that the part is misplaced. Although this does not necessarily lead to a minimum number of parts that need to be disconnected in order to get the assembly back to valid state, we are making the assumption that the user would rather correct their mistake as soon as possible instead of continuing to connect more parts albeit of the assembly being in an invalid state. Regardless, once the user disconnects any of the parts, we re-validate the assembly to see if the assembly has reached a valid state. Due to this, even if the user continues building while in an invalid state, it is still possible for them to correct the state with minimal number of disconnects.

If the assembly is in a valid state, we need to provide user instructions on what would be the next two assemblies they should join together. As mentioned earlier, due to the use of context-aware guidance, we cannot rely

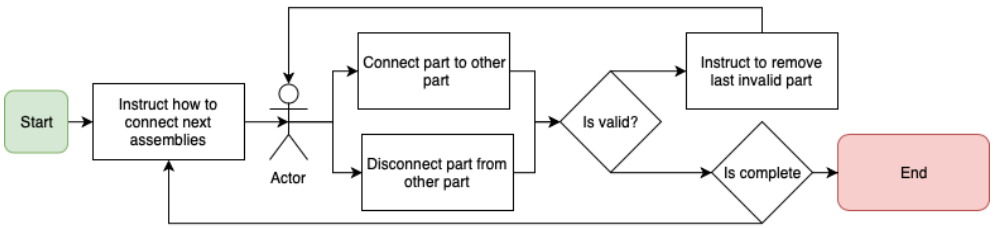


Figure 4.11: Assembly validation and guidance flowchart.

on predefined assembly instructions, so instead, we have to compute the next step dynamically. To do this, we use the one-to-one mappings between users' working parts and the parts in target assembly obtained during the validation. To find a suitable instruction for next step, we go through all pairs of assemblies user is working on, and check whether or not these two assemblies can be joined together while still keeping the assembly valid. Out of all of the valid pairs of assemblies, we use a simple heuristic for prioritisation where we give each of the users' assemblies a priority based on the highest priority of parts within that assembly and select the next step based on that. The priority of a single part, on the other hand, is provided by the target assembly where we provide a preferred order of attaching parts the contextual guidance tries to follow.

To visualise the next step, we rely on three kinds of visual cues. We highlight the smallest of the two assemblies that need to be connected with green colour. Additionally, we draw the smaller assembly as a ghost next to the larger assembly where it should be placed to, and we also show green arrows to highlight the connections that need to be formed to complete the assembly step. These visual cues were chosen mainly based on previous research such as [17, 33, 39].

4.9 Emergent physics behaviour

During the development of the application, we encountered some positive and negative side effects emerging from the PhysX physics engine that we did not initially consider.

On the positive side, the friction model of PhysX solves the case where we have a part wedged between two other parts during assembly (Figure 4.13) so well that we do not need to do anything manually to keep the part stably between the two other parts during the assembly. Friction and collisions between the washers of bolts and the parts the bolts are connected to also automatically limit the maximum rotation in threaded connections

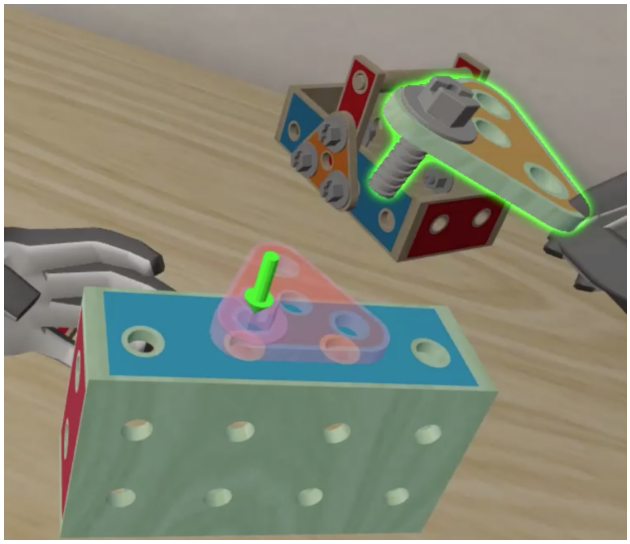


Figure 4.12: We use combination of green arrows, semi-transparent ghost parts, and a green highlight for displaying instructions for the user.

surprisingly well. There are still some cases where over-tightening threaded connections may cause some jitter, but these cases are in practice hard to encounter unless intentionally trying to break the system.

In addition to positive side effects, some negative side effects need to be taken into account to provide stable simulation. One of the most significant issues is that fast hand movements can sometimes cause parts to pass, or tunnel, through other parts resulting in the simulation reaching a state that would not be physically possible. In addition to the default discrete collision detection, Unity also supports two different methods for continuous collision detection (CCD) which should help in reducing the cases where objects pass through each other. When working on the hand-object interaction, we had enabled CCD to avoid the user being able to force parts through the table with rapid hand movements. However, later we noticed that counter-intuitively, having CCD enabled did not reduce the tunnelling between connected parts but surprisingly it made it worse. While we are not sure about the actual cause of the issue, we believe it is somehow related to long constraint chains in the virtual assemblies and at least partially addressable to the iterative nature of the constraint solving. Curiously, we were not able to fix this issue by decreasing the physics timestep or by increasing physics solver iteration counts when CCD was enabled. In the end, our solution to this problem is to keep using discrete collision detection to achieve more stable simulation. To fix the case where the user can force parts through the tabletop, we made the collider of the table thicker.

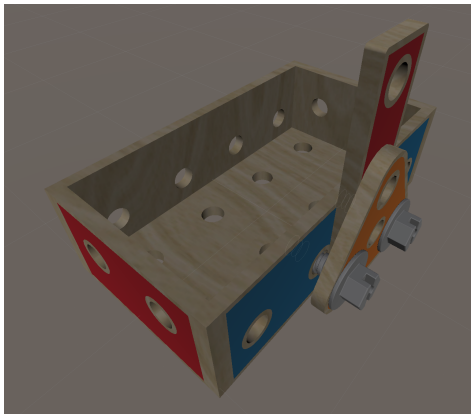


Figure 4.13: Red rectangle is wedged to its place by friction and collisions caused by the bolts clamping the rectangle between the orange triangle and the box.

Another unwanted side effect we observed is that some of the parts occasionally rotate by themselves when connected to other parts. This extra torque is caused by joint solver when it is trying to solve constraints with locked degrees of freedom and also due to lack of friction between the constrained parts. Even though we did not manage to eliminate the issue, we were able to significantly reduce this behaviour by introducing rotation damping force to the joints between objects. This damper is somewhat equivalent to having dynamic friction, and we believe that introducing artificial static friction could further alleviate this issue. The lack of friction between connected parts in shaft to hole connections also makes the assembly vulnerable to accidentally disconnecting parts as it is easy to induce small, unwanted motion to the assembly where momentum would also get applied to the parts that would not stop as there is no friction slowing them down.

4.10 Summary

In Chapter 1, we highlighted three specific areas of virtual mechanical assembly we wanted to improve on in our implementation. First, we wanted to provide a natural user interaction instead of commonly used user interfaces such as wand controllers. Our second goal was to create an assembly simulation that is both physically convincing and hand tracking friendly. Finally, we wanted to let the user complete the assembly tasks in any order rather than forcing them to follow specific step-by-step instructions.

To solve the first challenge, we used a Leap Motion sensor for hand track-

ing to give the user the possibility to use virtual hands for the assembly tasks. During the development, we experimented with multiple different approaches for grasping with virtual hands, and we concluded that for constraint-based assembly the best approach for grasping was to detect pinch gestures and use a spring-damper model to move the grasped part together with the hand. With this approach, the parts are softly connected to the hand, which allows them to adhere to the constraints used in the assembly simulation.

For the second challenge, we used a combination of real-time physics and constraint-based assembly, where we utilise physics joints to aid the user in connecting parts together. This way, parts can physically interact with each other until the user starts connecting them after which the constraints start guiding the parts to a spatial configuration that allows them to connect properly. In our application, we utilise the physics constraints for both guiding the parts to a favourable spatial configuration during the assembly operations and for keeping the parts connected.

Finally, to give the user the freedom of performing the assembly tasks in any order, we developed a custom assembly validation and guidance algorithm based on assembly matching. Every time the user connects or disconnects parts, we match the users' assembly to the target assembly to provide the user with context-aware assembly instructions, thus performing workflow validation and guidance at the same time.

Chapter 5

Evaluation

In this section, we evaluate our VR application for mechanical assembly training. To evaluate the usability of different aspects of the application, we conducted two small-scale user tests, one with university staff and one with people working on the engineering industry. We present the testing procedure and the results of the user tests in Section 5.1. Additionally, in Section 5.2, we provide a brief performance analysis of the application to show that it satisfies the real-time performance requirements in VR. Finally, in Section 5.3, we discuss some of the limitations in our application and also propose some possible solutions to overcome those limitations.

5.1 User testing

During the work on this thesis, we conducted two iterations of prototyping and user testing, and our goals for the user testing were twofold. In addition to validating the functionality of the application, we also wanted to study user preferences between voice commands and a more traditional user interface in VR. As there were almost two months between the two user tests for the application, we also took the opportunity to apply some of the findings from the first user test to improve the experience for the second test.

For the user tests, we used the following test procedure: First, we asked the testers to answer how familiar they are with VR and how often do they use it, and what was their first VR experience. After that, the testers were instructed to put on the HTC Vive VR headset and prepare themselves for the test. During the test, the users were asked to do three things. Start the assembly simulation, load an assembly task called *"Medium"*(Figure 4.4), and complete the assembly task following the visual instructions. Apart from using virtual hands to assemble the product, to perform the actions mentioned

above, the users were able to use either voice commands or graphical user interface based on their personal preference. Once the users completed the assembly or considered themselves to be done with the testing, they were instructed to fill out a questionnaire regarding the application.

To create the questionnaires for user tests, we selected multiple categorised questions from standardised usability questionnaires [4] and adapted those to better fit into different aspects of our application. Additionally, we introduced a question on how immersive the users perceived the application to be. Each of the questions was rated on a Likert scale from *"Strongly agree"* to *"Strongly disagree"* where the answers were mapped to numerical range from 1 to 5, correspondingly. Similarly to the way scores are interpreted in System Usability Scale (SUS) [12], the answers where then converted to numerical scores from 0 to 4 where 0 is the worst result and 4 is the best result by reducing the answer by one (answer - 1) if the question had a negative wording and taking away the number from five (5 - answer) if the question had a positive wording. The questions of the first and second test are provided in tables 5.1 and 5.2, correspondingly.

At the end of the questionnaire, the testers were also able to leave open feedback and any comments they had. Additionally, in the beginning of questionnaire of the first test, the users were asked to score the main features of the application based on what they considered significant (Table 5.3), and to also state their personal preference between voice commands and wrist UI.

5.1.1 First test

We organised our first user test with a group of 10 members from Mobile Cloud Computing-group to test the application. The group consisted of 8 male and 2 female participants with varying amounts of previous experience in VR, ranging from some people having almost no VR experience at all to those who used VR almost weekly. During the test, each of the users used the application from 20 to 45 minutes with average usage time at around 30 minutes.

Once the users had completed testing the application, they were tasked to fill a table rating each of the features based on what they feel is the importance of each feature (Table 5.3)

Looking at scores in Table 5.4, the users considered instructions to be important features of application as both *Highlighting the object* and *Instruction in the beginning* were valued by the users. On the contrary, both features related to voice commands were not considered as important. However, this may partially be affected by our implementation of voice commands as the users found the voice commands to be rather unclear as seen from the

Category	**Questions**
Immersion	0. Did you feel immersed in the VR environment
Effectiveness	1. Tasks can be performed in a straight forward manner using this system. 2. I could effectively complete the work using this system. 3. Overall, I am satisfied with the ease of completing the tasks in this system.
Efficiency	4. I found the system very cumbersome to use. 5. This system responds too slowly to inputs. 6. I have to spend too much time correcting things with this system. 7. Overall, I am satisfied with the amount of time it took to complete the tasks in this system.
Satisfaction	8. Overall reactions to the software satisfying. 9. Overall, I am satisfied with this system.
Detectability	10. The amount of the help information varies across the system. 11. The quality of the help information varies across the system. 12. My interaction with this system would be clear and understandable.
Discriminability	13. Organisation of information shown was very clear to understand.
Appropriateness	14. I found the various functions (voice, highlighting the object, wrist UI menu) in the system were well integrated. 15. I found using voice commands was well organised. 16. I found using wrist UI menu was well organised. 17. I found using highlighting objects was well organised. 18. I found showing animation of the goal was well organised.
Comprehensibility	19. I could understand and act on the information provided by this system. 20. I could understand how to use voice commands easily. 21. I could understand how to use wrist UI menu easily.
Guidance	22. Highlighting clarifies task to follow. 23. The organisation information of voice commands is clear. 24. The organisation information of visual UI is clear. 25. The organisation of the menus seems logical. 26. Overall, I am satisfied with the support information when completing the tasks.
Consistency	27. I thought there was too much inconsistency in this system.

Table 5.1: Categorised questions of the first user test.

Category	**Questions**
Effectiveness	1. I could effectively complete the work using this system. 2. Overall, I am satisfied with the ease of completing the tasks in this system.
Efficiency	3. I found the system very easy to use. 4. This system responds quickly to inputs. 5. Overall, I am satisfied with the amount of time it took to complete the tasks in this system.
Immersion	6. Did you feel immersed in the VR environment?
Satisfaction	7. Overall, I am satisfied with this system.
Comprehensibility	8. I could understand and act on the information provided by this system. 9. I could understand how to use voice commands easily. 10. I could understand how to use visual UI menu easily.
Guidance	11. Highlighting the object clarifies task to follow. 12. The organization information of voice commands is clear 13. The organization information of visual UI is clear. 14. The organization of the menus seems logical.

Table 5.2: Categorised questions of the second user test.

UI Property	Score
Voice Commands	
Wrist UI menu	
Highlighting the object	
Instruction showing the goal object in the beginning	
Voice instruction	
Total	**100**

Table 5.3: The features users we told to rate when answering the questionnaire during the first user test.

questionnaire results (Figure 5.1).

When looking at the responses for the rest of the questionnaire (Figures 5.1 and 5.2), it becomes clear that the users had significant issues while using the application. In particular, users did not feel that could effectively complete the work using the application and they found the application to be hard to use and the overall satisfaction towards the application was between 1 and 2 which is rather low. Users also found voice commands hard to use and this is also reflected in answers to questions 15, 20, and 23. On the other hand, answers related to the visual guidance (questions 17, 18, and 22) stand out as positive results, and in general users were happy with the provided assembly instructions and they were able to follow the guidance rather well. The disparity of the different aspects of the application is also reflected in the per-category scores in the form of wide confidence intervals.

When we asked for the users' preference between wrist UI and voice commands, 8 out of 10 users preferred the wrist UI. Even with the minimal set of voice commands, we observed that many of the users forgot the available voice commands soon after hearing the spoken instructions which forced them to use the wrist UI. For this reason, multiple users suggested having the voice commands visually available somewhere. Some users also had trouble talking with the test instructor while performing the task as sometimes voice recognition accidentally picked up commands.

During our user testing, we observed that multiple users struggled with virtual hands. Although couple users felt that hand tracking was surprisingly accurate, all of the users suffered from some difficulties with grasping or releasing objects. However, one of the test users found the assembly to be surprisingly easy after understanding the logic of grabbing and releasing objects. Other test subjects also felt like more training would have been required, especially regarding hand-object interaction and voice commands, to familiarise themselves with the system before being given the actual as-

UI Property	1	2	3	4*	5	6	7	8	9	10	Mean
Voice commands	10	0	10	19	25	10	5	15	15	20	12.9
Wrist UI	30	10	15	14	25	10	10	25	20	30	18.9
Highlighting the object	25	50	30	24	20	30	40	20	40	40	31.9
Instruction in the beginning	10	30	25	24	15	20	35	15	10	10	19.4
Voice instruction	25	10	20	19	15	30	10	25	15	0	16.9

Table 5.4: User-perceived importance of each of the main features. (*User 4 rated each of the properties between 0 and 100 so their scores were normalised to add up to total of 100.)

sembly task. Manipulation of nuts and bolts was also found to be clumsy due to lack of dexterous manipulation using fingers, and users highlighted that manipulating parts inside the box was especially challenging. Although some users stated that they used VR on a weekly basis, we did not observe any correlations between the users' previous experience with VR and their ability to interact with the application.

Another issue observed during testing was that it was easy to accidentally perform rapid hand movements and shake parts off the assembly. This was also apparent in the questionnaire results where over half of the testers felt like they had to spend too much time correcting things when using the system. One proposed solution for this issue was to group the assembly process into different phases and consider parts being fixed together as a single object once the phase has been finished. Although we initially considered this as well, we decided to leave it out to keep the application more physics-based. Furthermore, we did not want to group the assembly into different phases as we wanted to give the user the freedom of performing the assembly in an order they prefer. However, with this feedback in mind, we would reconsider this decision in future work. Another possible solution could be introducing additional artificial friction to the connections. This added friction would be especially useful when one of the attachments forming the connection is a threaded one as the threads themselves provide additional friction in real life. Furthermore, if the fingers and palms of virtual hands would collide with the virtual objects, the user would be able to hold two parts together with one hand while interacting with them using their other hand.

On the positive side, the users found the visual guidance to be helpful and the ability to make mistakes and recover from them was pointed out to be a positive experience. As a small improvement to guidance, it was suggested by one user that the visual guidance could also take into account the case where the parts are outside of the user's viewport by, for example, pointing towards the object with an arrow.

Based on the test results, we did the following changes to application: First, we changed the touch UI from a menu attached to users' wrist to a

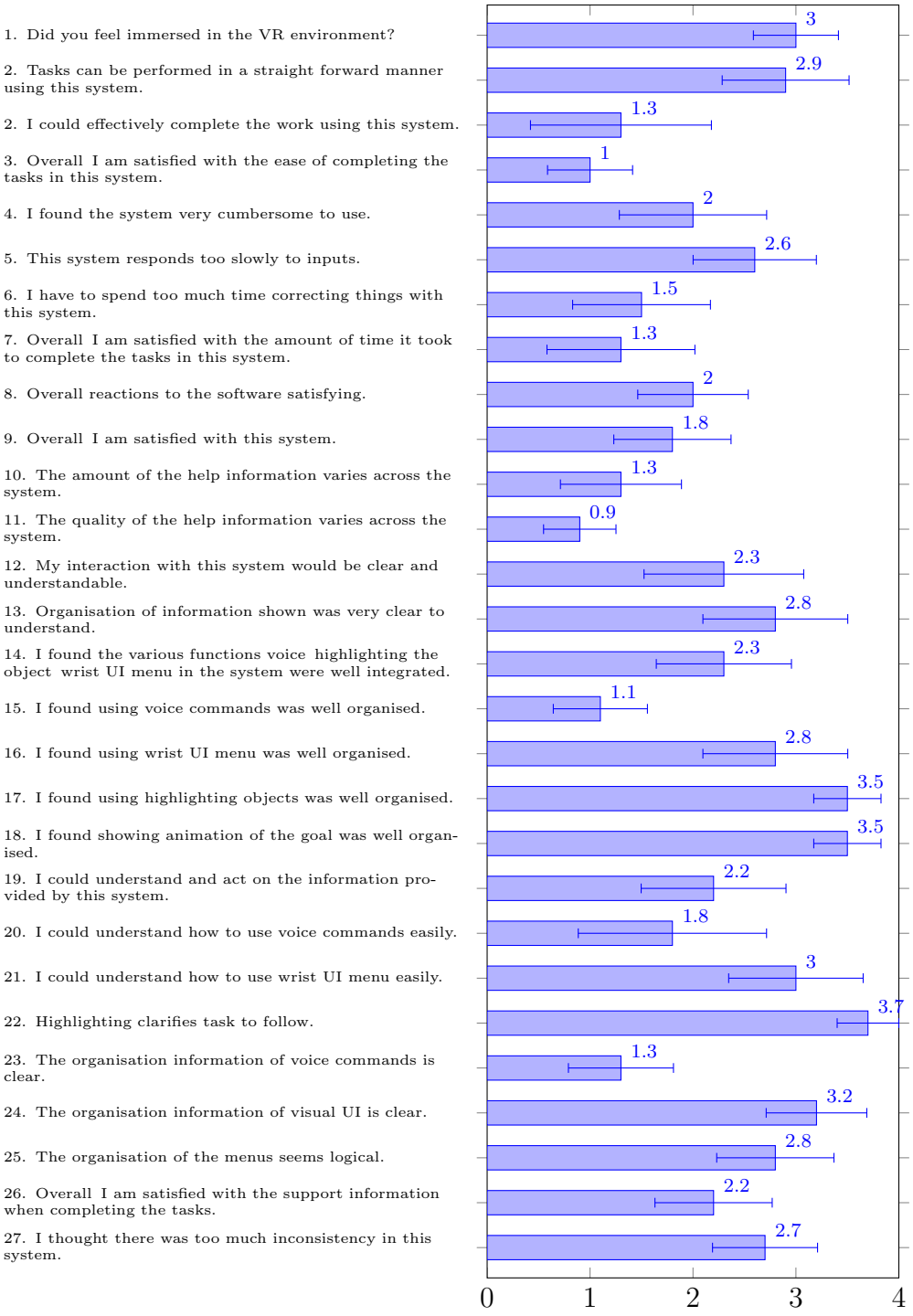


Figure 5.1: Mean scores of first test by question (the higher the better). Error bars represent 95% confidence interval.

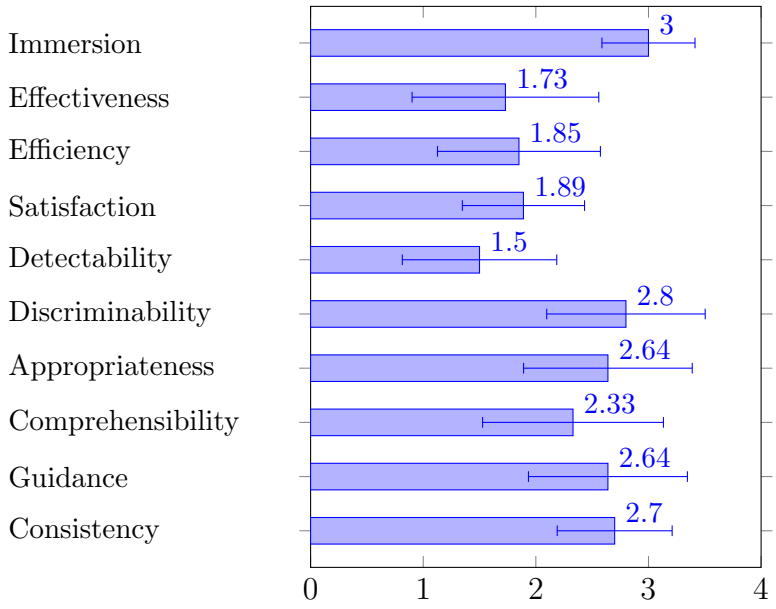


Figure 5.2: Mean scores of first test by category (the higher the better). Error bars represent 95% confidence interval.

menu floating in front of the user. Second, we added a new voice command, *voice*, that lists all the available voice commands. We also added visual feedback for *voice* and *list* commands by introducing a floating text panel that appears in front of the user and displays the spoken instructions as text. Although we would have wanted to improve the hand-object interaction as well, we decided to prioritise the user experience improvements instead due to time limitations.

5.1.2 Second test

After addressing some of the most pressing usability issues that came up in the first user test, we conducted a second user test with people working on the engineering industry. The participants of the test consisted of 2 females and 5 males, with most of the testers having over 19 years of experience in the industry. Each of the users had about 15 minutes for testing the application.

Before the test, the testers were shown a video clip of an earlier version of the application, but apart from that, none of them had seen or used the application beforehand. However, all of them had at least some previous experience in VR, and some even stated that they use VR more than once in a month. While we did not show the video for the users before the first user test, the test procedures were otherwise the same.

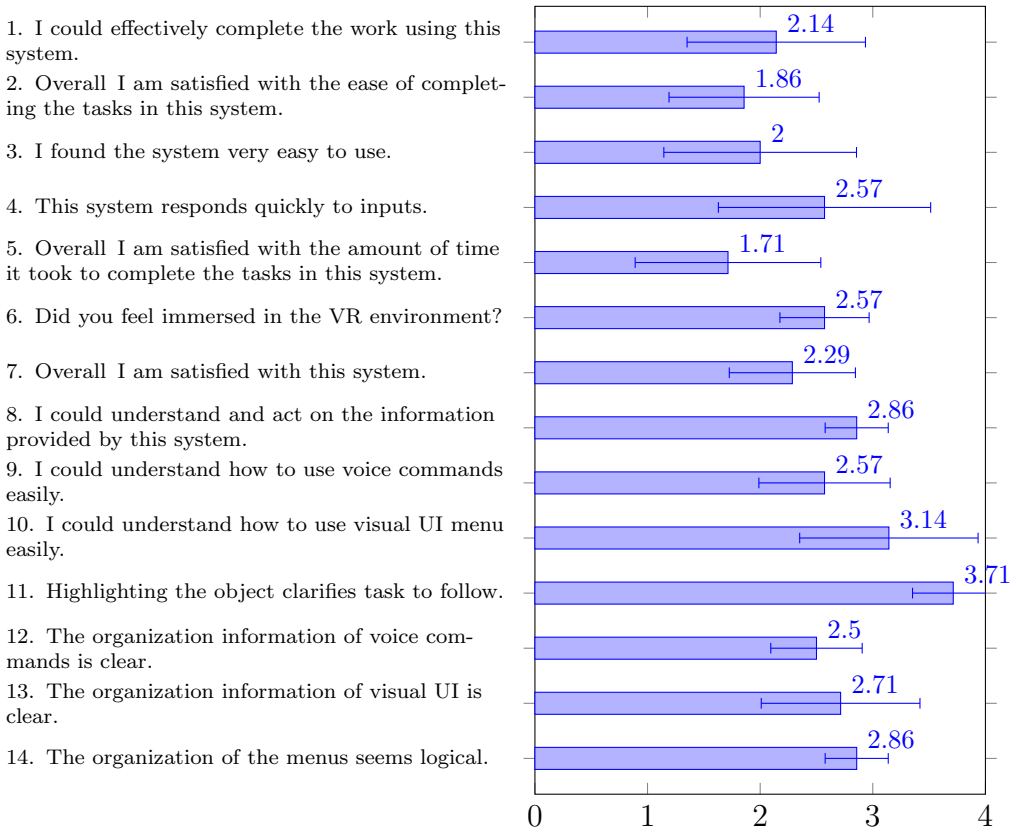


Figure 5.3: Mean scores of second test by question (higher is better). Error bars represent 95% confidence interval.

We did, however, streamline the questionnaire by reducing the number of questions to 14 and by rephrasing some of the questions. The scoring of the second test followed the one of the first test with the exception that all the questions in this test had a positive tone in them so the answers were simply converted to numerical values from 0 (strongly disagree) to 4 (strongly agree) thus providing comparable scores with the first test. The questions and the mean scores of the responses are provide in Figure 5.3.

Looking at the per-question average scores in Figure 5.3 it is clear that the users did not find the system to be very easy to use, and during the testing, none of the users were able to complete the assembly task given to them which led to a plenty of frustration during the testing. Similarly to the first user study, some of the users stated that more training could have helped them in using the application, and the general feeling when discussing with the users was that there is plenty of work required to make the application useful for real-life training. When discussing the issues after testing, one of the users

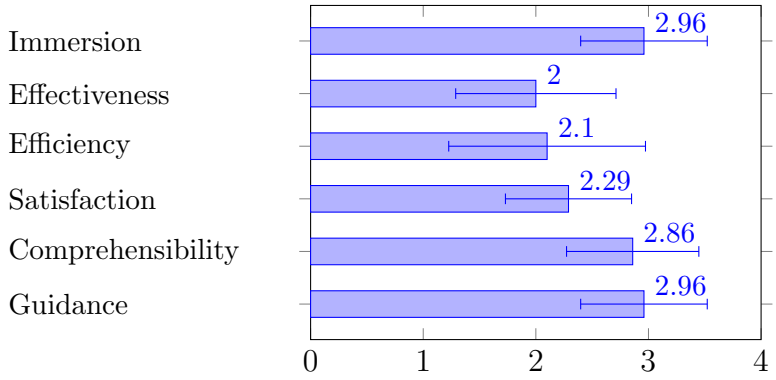


Figure 5.4: Mean scores of second test by category (the higher the better). Error bars represent 95% confidence interval.

mentioned that based on their previous experience with VR, they found VR controllers to be much easier to use compared to Leap Motion. Apart from the obvious usability issues, most of the per-question scores averaged between 2 and 3 leaning slightly towards the positive side but Similarly to the first user test, the users found the visual guidance to be helpful with a score of 3.71. The usability difficulties can also be seen in effectiveness, efficiency, and satisfaction-categories when looking the per-category average scores (Figure 5.4).

Although we did perform some improvements to the voice commands based on the findings of the first test, the users in the second test were not eager to use the voice commands and only one of the ten users tried using them before switching to the touch UI. We believe this was at least partially related to the fact that the second test was carried out in a small meeting room where other people were having discussions at the same time. Even though we did not get meaningful data regarding if the improvements to the voice commands were beneficial or not, we do think that the reluctance to using voice commands may also be reflected in real-life scenarios where the surrounding noise may be significant.

5.1.3 User test conclusions

The results of the user tests do not look very positive, and the reasons for that became clear from the open feedback left by the testers. Most of the users suffered from the clumsiness of the VR hands, which was at least partially caused by limitations of Leap Motion. While some users found the hands quite non-intuitive, other testers had issues with accidentally shaking parts of the assembly. It is, indeed, easy to accidentally move the virtual hands

rapidly as nothing is limiting the user from doing so due to lack of force feedback. Moreover, although we did try to minimise the possibility of Leap Motion losing track of hands by organising objects in the 3D environment so that as little head movement as possible would be required, the users still ended up in situations where that hand goes off the screen, becomes visible again, and is therefore moved back to its place rapidly.

In retrospect, we should have tried alternative approaches for hand-object interaction during the user test as well and in the second user test one of the testers even mentioned that they had found VR controllers to be easier to use in their earlier experience with VR. Some ideas for improving hand-object interaction are discussed in Section 6.1. On the other hand, some testers stated that once they learned how the system functioned, it was surprisingly easy to perform the assembly which suggests that with better initial training the users might have been able to have a better experience.

Although the user tests show somewhat disheartening results in terms of usability, there are multiple takeaways from the study for future projects. For instance, users were pleased with the visual, context-sensitive assembly instructions that were powered by our algorithm based on assembly matching. While some users wished for more strict workflows and clearer phases for assembly, most of the users found the contextual guidance working well and one of the users highlighted that they were satisfied with the ability to make a mistake.

5.2 Performance

We did not spend much time on performance optimisation due to them not being necessary. However, we did preemptively reduce the polygon counts of the 3D CAD models in Blender since most of the VR devices have to render everything twice, once for each eye, and VR headsets tend to provide bad experiences when frames per second (FPS) of the application does not match the FPS of the headset. During development and testing, we were able to keep stable 90 FPS on both of our test machines, a desktop gaming PC with Oculus Rift and a gaming laptop with HTC VIVE.

Both of the user tests were conducted using the aforementioned gaming laptop and during the testing we did not notice any stuttering or drops in the frame rate. Also, the average score for the responsiveness of the system was around 2.6 which suggests that the application works fast enough to satisfy the real-time requirements. When discussing about the reasons behind some of the lower responses affecting the average, some users highlighted that there was a noticeable delay in the speech recognition which was unrelated to the

Custom-built desktop PC	Asus Strix SCAR II laptfop
Oculus Rift	HTC VIVE
4-core Intel i7-6700 @ 4.00 GHz	6-core Intel i7-8750H @ 4.10 GHz
NVIDIA® GeForce GTX™ 1070	NVIDIA® GeForce RTX™ 2070
32GB DDR4	32GB DDR 4

Table 5.5: Hardware used for development and testing.

performance of the application itself but rather a limitation in the speech recognition library.

Most complex of our test assemblies, toolbox, contained 18 parts and we did not notice any performance issues during the assembly. It is worth mentioning, however, that if there is a second toolbox in the scene, the performance starts to suffer slightly, so if more complex assemblies are required, there are some performance improvements that can be carried out. A simple improvement would be adding logic to determine if some constraint can be considered as "frozen" and treat connected parts as one. In our test data, this would be applicable to screw connections as is, but a more complex logic could be applied to cases where a stack of objects is tightly pressed together by a screw connection. This approach would have the added benefit of reducing joint flexing caused by iterative joint solves trying to solve a chain of joints.

5.3 Limitations

For convenience, many of our implementation details rely on the assumption that there is always a single connection between two connected parts. While this assumption holds for all the parts in our toy set, we feel obligated to describe a possible workaround for this limitation as this assumption may not hold for all applications. A simple way of overcoming this limitation without having to change the behaviour of existing systems would be to treat these parts as special prefabricated assemblies, where a part is split to multiple artificial sub-parts that are joined together with non-breakable fixed connections.

By providing the parts and assemblies as unity assets we are effectively limiting the usability of the application to the assets that are included in the application when it was built. One way of introducing new parts and assemblies later would be to use *AssetBundles*, archive files that contain platform specific non-code assets such as prefabs and 3D meshes that can be loaded at run time. Since the asset bundles are authored inside Unity

editor, the application operator could still perform everything conveniently from the Unity editor. Similar approach has already been used by many games to give the users the power to add custom content. Alternatively, one could easily add support for loading the assembly tasks from files since the assembly loading has been designed with that in mind as it only relies on relative positions and orientations of the parts.

Chapter 6

Conclusions and future work

The goal of this thesis was to design and develop a virtual reality application for mechanical assembly training using readily available VR equipment and a commercial game engine. The main goal for the application was to improve on the lack of natural interaction in earlier virtual assembly applications by allowing the user to use their own hands in VR to build multi-part assemblies from virtual parts that can physically interact with each other. To achieve this, we combined the use of Leap Motion hand tracker with constraint-based assembly to create an immersive, hands-on virtual assembly application.

Although there is still plenty of space for improving the user experience, the application provides the functionality we set out to create, and we were aware beforehand that hand tracking technologies are still far from perfect and that the hand tracking could possibly cause issues. While the testers saw the hand-object interaction as a major source of frustration, some of the testers were able to successfully perform the assembly task after taking some time to familiarise themselves with the system.

Apart from the issues with the hand-object interaction, the users were generally satisfied with the context-aware instructions and they were stated to be easy to follow. Although our method for validation and guidance using assembly matching was geared towards context-awareness, we did not observe many users to go against the instructions so in that sense, it hard to judge how beneficial the context awareness is in the end. However, the ability to make mistakes and recover from them was highlighted as a positive feature by one of the testers. In addition solving the challenge of providing context-aware instructions, we also consider the other requirements we set for ourselves in Section 3.2 to be satisfied by the provided application.

In the end, while we did manage to satisfy the goals we set for ourselves and the users saw some potential in the application, the overall usability of the application leaves a plenty of room for improvements, and therefore

the application is not quite ready for real-life use cases. The hand tracking technology is not quite accurate enough for the kind of small-scale mechanical assembly present in the application, but we expect the tracking technology to be improved in the future with both hardware and software improvements.

6.1 Future work

Even though the final application provides an extensible platform for virtual assembly training, there are plenty of improvements to be done to improve the usability. Based on the feedback from the testers and our own experience during the development, here are some ideas we would like to work on the future:

Training mode Multiple users highlighted that a separate mode for familiarising themselves with the application would be helpful. Albeit that we see a training mode to a training application as a somewhat contradicting feature, we would be we willing to consider this as it was suggested by multiple users.

Hand-object interaction We feel that from the users perspective, the hand-object interaction is the feature that leaves the most room for improvement and this feeling was also backed up by the test results. Low accuracy of the hand tracking combined with the small scale of the objects being manipulated makes it difficult to perform dexterous manipulations such as operating screws or rotating objects with fingertips which significantly reduces the naturalness of the assembly. Although some studies, such as research by [31] have suggested that users might be more comfortable with less physical hand-object interactions, we observed that fastening nuts and bolts without dexterous manipulations was especially counter-intuitive as the user was required to twist their whole hand instead of just using their fingertips. We also acknowledge that the lack of physics collisions between virtual hands and the parts not only reduces the naturalness of the interaction but also results in some situations where the user can perform actions that would be impossible in the real world.

Force feedback Some previous studies, such as [25, 53] have highlighted that force feedback can significantly improve the virtual assembly performance. With this in mind, in spite of Leap Motion being a non-intrusive way implementing hand-object interaction, the user experience is likely to be significantly improved by introducing force feedback through a more complex, wearable hand-machine interface. For this reason, in the future, we would like to carry out another test with force feedback-enabled VR gloves. We believe this would likely allow us to revisit the decision of not having

collisions between virtual objects and virtual hands as the user would be able to get better feedback when interacting in the virtual environment. We would also like to conduct another study where the user would use either Oculus Touch controllers or HTC VIVE controllers to study how users feel about using controllers compared to natural hand interface provided by Leap Motion or VR gloves as controllers tend to have good tracking accuracy and haptic feedback. Oculus Touch controllers and Valve Knuckles controllers would be especially attractive options as both of them provide rather good finger tracking which would provide a good compromise between immersion and more explicit interaction through the use of buttons on the controllers.

Automatic feature detection While Unity editor provides a convenient and straightforward way of augmenting the imported CAD models with the information about attachments, some previous works, such as [19, 22, 43], have proposed model processing pipelines for automatically gathering the relevant contact information from the CAD models. Considering that a significant portion of the industrial parts is designed using CAD applications, integrating automatic contact identification as part of the model import pipeline would reduce the manual labour required for setting up the parts in Unity to practically none. Although this would have no impact on the end-user using the application for training, it would offer some time savings for the person in charge of setting up the parts for the simulation.

Maintenance training Considering that assembled products may also require maintenance during their lifespan, we would like to expand the application to support training of maintenance tasks to broaden its usability. Due to connecting and disconnecting of parts already being possible, this expansion would mostly involve changing the logic used for displaying the step-by-step instructions to the user while keeping other systems untouched.

Alternative physics engine Although PhysX is already bundled into Unity, we would like to try other physics engines as well to see if alternative solutions would provide, for example, more rigid joints or less unwanted side effects. MuJoCo [28] would be especially attracting option as it has been shown to be both faster and more accurate for multi-joint dynamics simulations compared to the physics engines targeting games [18]. We believe this would be a rather easy experiment to conduct since MuJoCo already provides a plugin for Unity integration,

6.2 Final thoughts

With the ongoing improvements in both usability and affordability of VR hardware, it seems likely that the use of VR in industrial applications such

as assembly training will continue to grow in the future. VR training is particularly attractive for situations where traditional training is dangerous or expensive, and since previous research has observed situations where VR training has been more effective than traditional training, we would expect VR training to become even more widespread.

Bibliography

[1] Abidi, M. H., Ahmad, A., El-Tamimi, A. M., and Al-Ahmari, A. M. Development and evaluation of a virtual assembly trainer. *Proceedings of the Human Factors and Ergonomics Society*, October (2012), 2560–2564.

[2] Adams, R. J., Klowden, D., and Hannaford, B. Virtual Training for a Manual Assembly Task. *Haptics-e 2*, 2 (2001), 1–7.

[3] Agrawala, M., Phan, D., Heiser, J., Haymaker, J., Klingner, J., Hanrahan, P., and Tversky, B. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics 22*, 3 (2003), 828.

[4] Assila, A., Ezzedine, H., et al. Standardized usability questionnaires: Features and quality focus. *Electronic Journal of Computer Science and Information Technology: eJCIST 6*, 1 (2016).

[5] Aziz, E.-S. S., Chang, Y., Esche, S. K., and Chassapis, C. Virtual Mechanical Assembly Training Based on a 3D Game Engine. *Computer-Aided Design and Applications 12*, 2 (mar 2015), 119–134.

[6] Behandish, M., and Ilies, H. T. Peg-in-Hole Revisited: A Generic Force Model for Haptic Assembly. *Journal of Computing and Information Science in Engineering 15*, 4 (aug 2015), 041004.

[7] blender.org - Home of the Blender project - Free and Open 3D Creation Software. `https://www.blender.org/`. Accessed: 2019-01-20.

[8] Boritz, J., and Booth, K. A study of interactive 6 DOF docking in a computerised virtual environment. In *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180)* (2004), IEEE Comput. Soc, pp. 139–146.

[9] Borst, C. W., and Indugula, A. P. Realistic virtual grasping. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*

(Washington, DC, USA, 2005), VR '05, IEEE Computer Society, pp. 91–98, 320.

[10] Borst, C. W., and Indugula, A. P. A spring model for whole-hand virtual grasping. *Presence: Teleoperators and Virtual Environments 15*, 1 (2006), 47–61.

[11] Boud, A., Haniff, D., Baber, C., and Steiner, S. Virtual reality and augmented reality as a training tool for assembly tasks. In *1999 IEEE International Conference on Information Visualization (Cat. No. PR00210)* (1999), IEEE Comput. Soc, pp. 32–36.

[12] Brooke, J., et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry 189*, 194 (1996), 4–7.

[13] CaptoGlove® - Virtual Reality & smart glove VR AR PC mobile gaming. `https://www.captoglove.com`. Accessed: 2019-06-08.

[14] Claeys, A., Hoedt, S., Van Landeghem, H., and Cottyn, J. Generic Model for Managing Context-Aware Assembly Instructions. *IFAC-PapersOnLine 49*, 12 (2016), 1181–1186.

[15] Controls | Wii | Nintendo. `https://www.nintendo.co.uk/Wii/Wii-mini/Controls/Controls-726560.html`. Accessed: 2019-07-06.

[16] de Mello, L. S. H., and Sanderson, A. C. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation 7*, 2 (Apr 1991), 228–240.

[17] Dwivedi, P., Cline, D., Joe, C., and Etemadpour, R. Manual assembly training in virtual environments. *Proceedings - IEEE 18th International Conference on Advanced Learning Technologies, ICALT 2018*, February 2019 (2018), 395–399.

[18] Erez, T., Tassa, Y., and Todorov, E. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (May 2015), pp. 4397–4404.

[19] Gonzalez-Badillo, G., Medellin-Castillo, H., Ritchie, J., Garbaya, S., and Lim, T. The development of a physics and constraint-based haptic virtual assembly system. *Assembly Automation 34*, 1 (2014), 41–55.

[20] Hu, W., and Zhang, X. A rapid development method of virtual assembly experiments based on 3d game engine. In *2nd International Conference on Electronic |& Mechanical Engineering and Information Technology* (2012/09), Atlantis Press.

[21] Höll, M., Oberweger, M., Arth, C., and Lepetit, V. Efficient Physics-Based Implementation for Realistic Hand-Object Interaction in Virtual Reality. *25th IEEE Conference on Virtual Reality and 3D User Interfaces, VR 2018 - Proceedings* (2018), 175–182.

[22] Iacob, R., Mitrouchev, P., and Léon, J. C. Contact identification for assembly-disassembly simulation with a haptic device. *Visual Computer 24*, 11 (jan 2008), 973–979.

[23] Khuong, B. M., Kiyokawa, K., Miller, A., La Viola, J. J., Mashita, T., and Takemura, H. The effectiveness of an ar-based context-aware assembly support system in object assembly. In *2014 IEEE Virtual Reality (VR)* (March 2014), pp. 57–62.

[24] Leap Motion. `https://www.leapmotion.com/`. Accessed: 2019-06-08.

[25] Lim, T., Ritchie, J. M., Dewar, R. G., Corney, J. R., Wilkinson, P., Calis, M., Desmulliez, M., and Fang, J. J. Factors affecting user performance in haptic assembly. *Virtual Reality 11*, 4 (2007), 241–252.

[26] Liu, H., Zhang, Z., Xie, X., Zhu, Y., Liu, Y., Wang, Y., and Zhu, S.-C. High-fidelity grasping in virtual reality using a glove-based system. In *International Conference on Robotics and Automation (ICRA)* (2019).

[27] Lu, X., Qi, Y., Zhou, T., and Yao, X. Constraint-based virtual assembly training system for aircraft engine. *Advances in Intelligent and Soft Computing 141 AISC* (2012), 105–112.

[28] Mujoco - advanced physics simulation. `http://www.mujoco.org/index.html`. Accessed: 2019-09-21.

[29] Murray, N., and Fernando, T. An immersive assembly and maintenance simulation environment. In *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications* (Oct 2004), pp. 159–166.

[30] Nasim, K., and Kim, Y. J. Physics-based interactive virtual grasping. In *Proceedings of HCI Korea* (South Korea, 2016), HCIK '16, Hanbit Media, Inc., pp. 114–120.

[31] Nasim, K., and Kim, Y. J. Physics-based assistive grasping for robust object manipulation in virtual reality. *Computer Animation and Virtual Worlds 29*, 3-4 (2018), 1–13.

[32] Nissen, M. S. Towards a Simpler Stiffer and more Stable Spring. `https://www.gamedev.net/articles/programming/math-and-physics/towards-a-simpler-stiffer-and-more-stable-spring-r3227/`. Accessed: 2019-07-13.

[33] Noghabaei, M., Asadi, K., and Han, K. Virtual manipulation in an immersive virtual environment: Simulation of virtual assembly. *CoRR abs/1902.05099* (2019).

[34] Object Interaction Part 2: Locomotion and Held Objects. `https://developer.oculus.com/blog/object-interaction-part-2-locomotion-and-held-objects/`. Accessed: 2019-06-08.

[35] Oculus Rift: VR Headset for VR Ready PCs | Oculus. `https://www.oculus.com/rift/`. Accessed: 2019-01-02.

[36] Ott, R., Vexo, F., and Thalmann, D. Two-handed haptic manipulation for CAD and VR applications. *Computer-Aided Design and Applications 7*, 1 (2010), 125–138.

[37] Pham, T.-A., and Xiao, Y. Unsupervised workflow extraction from first-person video of mechanical assembly. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications* (New York, NY, USA, 2018), HotMobile '18, ACM, pp. 31–36.

[38] PlayStation Move motion controller | More Ways to Play | PlayStation. `https://www.playstation.com/en-gb/explore/accessories/playstation-move-motion-controller/`. Accessed: 2019-07-06.

[39] Sanna, A., Manuri, F., Piumatti, G., Paravati, G., Lamberti, F., and Pezzolla, P. A flexible ar-based training system for industrial maintenance. In *Augmented and Virtual Reality* (Cham, 2015), L. T. De Paolis and A. Mongelli, Eds., Springer International Publishing, pp. 314–331.

[40] Seth, A., Vance, J. M., and Oliver, J. H. Combining Dynamic Modeling With Geometric Constraint Management to Support Low Clearance Virtual Manual Assembly. *Journal of Mechanical Design 132*, 8 (2010), 081002.

[41] Seth, A., Vance, J. M., and Oliver, J. H. Virtual reality for assembly methods prototyping: a review. *Virtual Reality 15*, 1 (mar 2011), 5–20.

[42] Shapr: 3D modeling CAD for iPad. `https://www.shapr3d.com/`. Accessed: 2019-01-20.

[43] Sheng, B., Yin, X., Zhang, C., Zhao, F., Fang, Z., and Xiao, Z. A rapid virtual assembly approach for 3D models of production line equipment based on the smart recognition of assembly features. *Journal of Ambient Intelligence and Humanized Computing 10*, 3 (2019), 1257–1270.

[44] Sherman, W. R., and Craig, A. B. *Understanding Virtual Reality: Interface, Application, and Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[45] Shiratuddin, M., and Thabet, W. Utilizing a 3D game engine to develop a virtual design review system. *Electronic Journal of Information Technology in Construction 16*, August 2010 (2011), 39–68.

[46] Tching, L., Dumont, G., and Perret, J. Interactive simulation of cad models assemblies using virtual constraint guidance. *International Journal on Interactive Design and Manufacturing (IJIDeM) 4*, 2 (May 2010), 95–102.

[47] Unity. `https://unity3d.com/`. Accessed: 2019-01-02.

[48] Virtuose™ 3D Desktop - HAPTION SA. `https://www.haption.com/en/products-en/virtuose-3d-desktop-en.html`. Accessed: 2019-07-06.

[49] VIVE™ | Discover Virtual Reality Beyond Imagination. `https://www.vive.com/eu/`. Accessed: 2019-05-06.

[50] Wang, Y., Jayaram, U., Jayaram, S., and Imtiyaz, S. Methods and algorithms for constraint-based virtual assembly. *Virtual Reality 6*, 4 (Aug 2003), 229–243.

[51] Wang, Z.-R., Wang, P., Xing, L., Mei, L.-P., Zhao, J., and Zhang, T. Leap Motion-based virtual reality training for improving motor functional recovery of upper limbs and neural reorganization in subacute stroke patients. *Neural regeneration research 12*, 11 (2017), 1823–1831.

[52] Weber, P., Rueckert, E., Calandra, R., Peters, J., and Beckerle, P. A low-cost sensor glove with vibrotactile feedback and multiple finger joint and hand motion sensing for human-robot interaction. *25th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2016* (2016), 99–104.

[53] Xia, P., Lopes, A. M., and Restivo, M. T. A review of virtual reality and haptics for product assembly (part 1): Rigid parts. *Assembly Automation 33*, 1 (2013), 68–77.

[54] Zhang, Y., Fernando, T., Xiao, H., and Travis, A. R. L. Evaluation of Auditory and Visual Feedback on Task Performance in a Virtual Assembly Environment. *Presence: Teleoperators and Virtual Environments 15*, 6 (dec 2006), 613–626.